3 Grammars

Syllabus

Grammar introduction - Types of grammar - Context free grammars and languages - Derivations and languages - Ambiguity - Relationship between derivation and derivation trees - Simplification of CFG - Elimination of useless symbols - Unit productions - Null productions - Greiback normal form - Chomsky normal form - Problems related to CNF and GNF.

Contents

3.1 Types of Grammar

The Chomsky hierarchy defines following types of grammar -

1. Type 3 Grammar :

- This type of grammar is also called as **regular grammar.**
- This grammar generates the regular language.
- The production rule for this kind of grammar is as follows -

  A->a

  and

  A->aB
- This grammar can be modeled using finite automata.

2. Type 2 Grammar :

- This type of grammar is also called as **context free grammar.**
- This grammar generates the context free languages.
- The production rule for this kind of grammar is as follows -

  A->r

  Where A is a non terminal and r is a string of non terminals and terminals.
- This grammar can be modeled using push down automata.

3. Type 1 Grammar :

- This type of grammar is also called as **context sensitive grammar.**
- This grammar generates the context sensitive languages.
- The production rule for this kind of grammar is as follows - $\alpha A\beta \rightarrow \alpha \gamma \beta$ with A as non terminal and α, β, γ are string of terminal and non terminals.
- This grammar can be modeled using linear bounded automata.

4. Type 0 Grammar :

- This type of grammar generates **recursively enumerable languages.**
- The production rule for this kind of grammar is as follows -

  $\alpha \rightarrow \beta$
- This grammar can be modeled using Turing Machine.

3.2 Context Free Grammars and Languages

Definition :

The context free grammar can be formally defined as a set denoted by $G = (V, T, P, S)$ where V and T are set of non-terminals and terminals respectively. P is set of production rules, where each production rule is in the form of

non-terminal → non-terminals

or non-terminal → terminals

S is a start symbol.

For example,

$$P = \{ S \rightarrow S + S$$
$$S \rightarrow S * S$$
$$S \rightarrow (S)$$
$$S \rightarrow 4 \}$$

If the language is 4 + 4 * 4 then we can use the production rules given by P. The start symbol is S. The number of non-terminals in the rules P is one and the only non terminal i.e. S. The terminals are +, *, (,) and 4.

We are using following conventions.

1. The capital letters are used to denote the non-terminals.
2. The lower case letters are used to denote the terminals.

Example : The formation of production rules for checking syntax of any English statement is

SENTENCE → NOUN VERB

NOUN → Rama / Seeta / Gopal

VERB → goes / writes / sings

Thus if want to derive a string "Rama sings" then we can follow the above rules.

3.3 Derivations and Languages

AU : May-10,11,12, Dec.-11,14, Marks 16

The production rules are used to derive certain strings. We will now formally define the language generated by grammar G = (V, T, P, S). The generation of language using specific rules is called derivation.

Definition :

Let G = (V, T, P, S) be the context free grammar. If $A \rightarrow \beta$ is a production of P and α and γ are any strings from non-terminals or terminals i.e. in (VUT)* then $\alpha A \gamma \rightarrow \alpha \beta \gamma$.

Suppose $a_1, a_2, a_3, \ldots a_m$ are strings in (VUT)*, $m \geq 1$.

$$a_1 \Rightarrow a_2$$

$$a_2 \Rightarrow a_3$$

$$\vdots$$

$$a_{m-1} \Rightarrow a_m$$

Then we can say that $a_1 \underset{G}{\overset{*}{\Rightarrow}} a_m$

The language generated by G is denoted by L(G). The language L is called **context free language** CFL if L(G) is for CFG.

For example

G = ({S, B} {a, b}, {S $\rightarrow$ a B b, S)

B $\rightarrow$ bbb }

is a grammar. Then we can derive a string abbbb as -

i) We will first start from start symbol S

$S \Rightarrow a\ B\ b$

ii) Then we will replace B by bbb

$S \Rightarrow a\ B\ b \Rightarrow a\ bbbb$

Thus we can obtain the desired language L by certain rules. The language L can be described as a language which starts with letter *a* and having 4 b's following. Let us solve some examples for derivation of CFG.

Example for Understanding

Example 3.3.1 *Construct the CFG for the language having any number of a's over the set* $\Sigma = \{a\}$.

Solution : As we know the regular expression for above mentioned language is

r.e. = a^*

Let us build the production rules for the same,

$S \rightarrow a\ S$ rule 1

$S \rightarrow \varepsilon$ rule 2

Now if want "*aaaaa*" string to be derived we can start with start symbols.

S

a S

a a S ∵ rule 1

a a a S rule 1

a a a a S rule 1

a a a a a S rule 1

a a a a a ε rule 2

$= a\ a\ a\ a\ a$

The r.e. = a^* suggest a set of {ε, *a*, *aa*, *aaa*, ...}. We can have a null string simply because S is a start symbol, and rule 2 gives $S \rightarrow \varepsilon$.

Solved Examples

Example 3.3.2 *Try to recognize the language L for given CFG.*

$G = [\{S\}, \{a, b\}, P, \{S\}]$

$$\text{where } P = \begin{Bmatrix} S \rightarrow aSb \\ S \rightarrow ab \end{Bmatrix}$$

Solution : Since $S \rightarrow a\ S\ b \mid a\ b$ is a rule. | indicates the 'or' operator.

$S \rightarrow a\ S\ b$

If this rule can be recursively applied then,

S
a S b
↓
a a S b b
↓
a a a S b b b

and if finally we can put $S \rightarrow ab$ then it becomes *aaaa* bbbb. Thus we can have any number of *a*'s first then equal number of b's following it. Hence we can guess the language as $\{L = a^n\ b^n \text{ where } n \geq 1\}$. The only way to recognize the language is to try out various strings from the given production rules. Simply by observing the derived strings, one can find out the language getting generated from given CFG.

Example 3.3.3 *Construct the CFG for the regular expression* $(0+1)^*$

Solution : The CFG can be given by,

$$P = \{S \rightarrow 0S \mid 1S$$
$$S \rightarrow \varepsilon \quad \}$$

The rules are in combination of 0's and 1's with the start symbol. Since $(0+1)^*$ indicates {ε, 0, 1, 01, 10, 00, 11, ...} in this set ε is a string. So in the rules we can set the rule $S \rightarrow \varepsilon$.

Example 3.3.4 *Construct a grammar for the language containing strings of atleast two a's.*

Solution : Let $G = (V, T, P, S)$

where $V = \{S, A\}$

$T = \{a, b\}$

$P = \{S \rightarrow Aa\ Aa\ A$

$A \rightarrow aA \mid bA \mid \varepsilon \}$

The rule $S \rightarrow AaAaA$ is something in which the two *a*'s are maintained since at least two *a*'s should be there in the strings. And $A \rightarrow aA \mid bA \mid \varepsilon$ gives any combination of *a*'s and b's i.e. this rules gives the strings of $(a + b)^*$.

Thus the logic for this example will be

(any thing) *a* (any thing) *a* (any thing)

So before *a* or after *a* there could be any combination of *a*'s and b's.

Example 3.3.5 *Construct a grammar generating* $L = w\,c\,w^T$ *where* $w \in \{a, b\}^*$

Solution : The strings which can be generated for given L is {*aacaa*, bcb, *abcba*, *bacab* ...}

The grammar could be

$$S \rightarrow a\ S\ a$$
$$S \rightarrow b\ S\ b$$
$$S \rightarrow c$$

Since the language $L = w\,c\,w^T$ where $w \in (a + b)^*$

Hence $S \rightarrow a\ S\ a$ or $S \rightarrow b\ S\ b$. The string abcba can be generated from given production rules as

S

$a\ S\ a$

$a\ b\ S\ b$

$a\ b\ c\ b\ a$

a	b	c	b	a

a	b	c	b	a

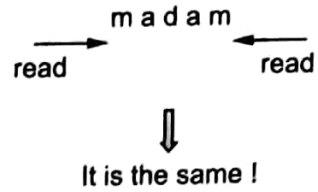
a	b	c	b	a

Thus any of this kind of string could be derived from the given production rules.

Example 3.3.6 *Construct CFG for the language L which has all the strings which are all palindrome over* $\Sigma = \{a, b\}$.

Solution : As we know the strings are palindrome if they posses same alphabets from forward as well as from backward.

For example, the string "madam" is a palindrome because



Since the language L is over Σ = {*a*, b}. We want the production rules to be build *a*'s and b's. As ε can be the palindrome, *a* can be palindrome even b can be palindrome. So we can write the production rules as

$$G = (\{S\}, \{a, b\}, P, S)$$

P can be

$S \rightarrow a\ S\ a$

$S \rightarrow b\ S\ b$

$S \rightarrow a$

$S \rightarrow b$

$S \rightarrow \varepsilon$

The string abaaba can be derived as

S

$a\ S\ a$ a b a a b a

$a\ b\ S\ b\ a$ a b a a b a

$a\ b\ a\ S\ a\ b\ a$ a b a a b a

$a\ b\ a\ \varepsilon\ a\ b\ a$ a b a a b a

$a\ b\ a\ a\ b\ a$ a b a a b a

which is a palindrome.

Example 3.3.7 *Construct CFG which consists of all the strings having atleast one occurrence of 000.*

Solution : The CFG for this language can be equivalent to r.e. (any thing) (000) (anything)

Thus r.e. = $(0+1)^*000\ (0+1)^*$

Let us build the production rules as

$S \rightarrow ATA$

$A \rightarrow 0A \mid 1A \mid \varepsilon$

$T \rightarrow 000$

Example 3.3.8 *Construct CFG for the language in which there are no consecutive b's, the strings may or may not have consecutive a's.*

Solution : $S \rightarrow a\,S \mid bA \mid a \mid b \mid \varepsilon$

$A \rightarrow a\,S \mid a \mid \varepsilon$

In the above rules, there is no condition on occurrence of *a*'s. But no consecutive b's are allowed. Note that in the rule $S \rightarrow bA$, and A gives all the strings which are starting with letter *a*.

Thus $G = (\{S, A\}, \{a, b\}, P, S)$

Let us derive the string

Derivation	Input	Productions
S		
a S	*a a* b *a* b	S → *a* S
a a S	*a a* b *a* b	S → *a* S
a a b A	*a a* b *a* b	S → b A
a a b *a* S	*a a* b *a* b	A → *a* S
a a b *a* b	*a a* b *a* b	S → b

Example 3.3.9 *Recognize the context free language for the given CFG.*

$S \rightarrow a\,B \mid b\,A$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

Solution : To find the language denoted by given CFG we try to derive the rules and get various strings. Then after observing those strings we come to know, which language it is denoting. Let us rewrite all those rules and number them

$S \rightarrow a\,B$	rule 1
$S \rightarrow b\,A$	rule 2
$A \rightarrow a$	rule 3
$A \rightarrow a\,S$	rule 4
$A \rightarrow b\,A\,A$	rule 5
$B \rightarrow b$	rule 6
$B \rightarrow b\,S$	rule 7
$B \rightarrow a\,B\,B$	rule 8

Now let us apply the rules randomly starting with start symbol.

S	
a B	rule 1
a a B B	rule 8
a a b S B	rule 7
a a b b A B	rule 2
a a b b *a* B	rule 3
a a b b *a* b S	rule 7
a a b b *a* b b A	rule 2
a a b b *a* b b *a*	rule 3

We have got some string as "*aa*bb*a*bb*a*". Let us try out something else.

S	
b A	rule 2
b b A A	rule 5
b b *a* S A	rule 4
b b *a a* B A	rule 1
b b *a a* b A	rule 6
b b *a a* b *a*	rule 3

Now we have got "bb*aa*b*a*" such a string !

Thus by obtaining more and more strings by applying more and more rules randomly will help us to find out what language it indicates. Observe the strings generated carefully, we can draw a conclusion as these strings contain equal number of *a*'s and equal number of b's. Even you can try out various rules to obtain some more strings. And see that it is a language L containing all the strings having equal number of *a*'s and b's.

Example 3.3.10 *Construct CFG for the language containing atleast one occurrence of double a.*

Solution : The CFG can be built with a logic as : one rule we will built for double *a* i.e. $A \rightarrow aa$.

And the other rule we will built for any number of *a*'s and b's in any combination.

i.e. $B \rightarrow aB \mid bB \mid \varepsilon$ which is always equivalent to $(a + b)^*$ (you can note it as golden rule !)

If we combine both of these rules we can get the desired context free grammar.

$$S \rightarrow BAB \Rightarrow (\text{anything}) \begin{pmatrix} \text{one occurrence} \\ \text{of double } a \end{pmatrix} (\text{anything})$$

$A \rightarrow aa$

$B \rightarrow aB \mid bB \mid \varepsilon$

Thus the CFG contains V = {A, B, S} T = {a, b}. The start symbol is S.

Let us derive "abaab"

S		
BAB		
aBAB	abaab	$B \rightarrow a\ B$
abBAB	abbaab	$B \rightarrow bB$
abAB	abbaab	$B \rightarrow \varepsilon$
abaaB	abbaab	$A \rightarrow a\ a$
abaabB	abbaabb	$B \rightarrow bB$
abaabε	abbaabb	$B \rightarrow \varepsilon$
= abaab	abbaabb	

Example 3.3.11 *Construct CFG for the language containing all the strings of different first and last symbols over* $\Sigma = \{0,1\}$.

Solution : Since the problem statement says as if the string starts with 0 it should end with 1 or if the string starts with 1 it should end with 0.

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

$A \rightarrow 0\ A \mid 1\ A \mid \varepsilon$

Thus clearly, in the above CFG different start and end symbols are maintained. The non terminal $A \rightarrow 0A \mid 1A \mid \varepsilon$ indicates $(0 + 1)^*$. Thus the given CFG is equivalent to the regular expression $[0(0+1)^*1+1(0+1)^*0]$

Example 3.3.12 *Construct the CFG for the language* $L = a^n b^{2n}$ *where* $n \geq 1$.

Solution : As in some previous example we have seen the case of $a^n b^n$. Now the number of b's are doubled. So we can write

$S \rightarrow aSbb \mid abb$

It is as simple as this !

Example 3.3.13 *Construct the production rules for defining a language* $L = \{a^x b^y \mid x \neq y\}$.

Solution : This is the language in which all the a's must appear before all the b's. But total number of a's must not be equal to total number of b's. Either number of a's must be equal to number of b's or number of b's must be equal to number of a's. The rule

$$S \rightarrow aSb$$

gives us equal number of a's must be followed by equal number of b's. But according to problem statement we need more number of a's either or more number of b's. Hence the required production rules can be -

$$s \rightarrow aSb \mid R1 \mid R2$$

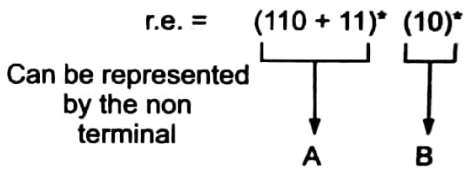
$$R1 \rightarrow aR1 \mid a$$

$$R2 \rightarrow bR2 \mid b$$

where S is a start symbol.

Example 3.3.14 *Find the CFG for the regular expression (110 + 11)*(10)*.*

Solution : Let



If we assume S as start symbol then,

$$S \rightarrow AB$$

Now for each non-terminal A and B we can define production rules as

$A \rightarrow 110A \mid 11\ A \mid \varepsilon$

$B \rightarrow 10\ B \mid \varepsilon$

Hence, finally the CFG for given r.e. can be

$S \rightarrow AB$

$A \rightarrow 110A \mid 11A \mid \varepsilon$

$B \rightarrow 10B \mid \varepsilon$

Example 3.3.15 *Build a CFG for generating the integers.*

Solution : The integer is any numerical value without decimal place which can be either signed or unsigned. Hence the CFG can be -

$$S \rightarrow GI$$

$$G \rightarrow + \mid -$$

$$I \rightarrow DI \mid D$$

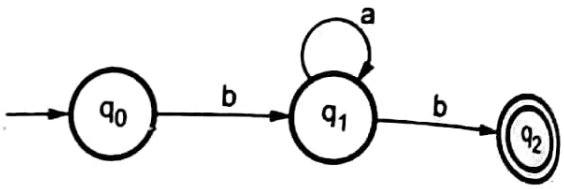
$$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \ldots \mid 9$$

}

S is a start symbol.

This is a right linear grammar because the non-terminal appears at the right end of the rule.

iv) The DFA will be



Example 3.3.20 *Construct a CFG for the set* $\{a^i b^j c^k | i \neq j \text{ or } j \neq k\}$ *As there are two cases* $i \neq j$ *or* $j \neq k$.

AU : May-11, Marks 6

Solution : Hence the production rules will be

$S \rightarrow R1 \mid R2$

$R1 \rightarrow aAbbBC \mid aaAbBC$

$R2 \rightarrow AbBccC \mid AbbBcC$

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow bB \mid \varepsilon$

$C \rightarrow cC \mid \varepsilon$

Example 3.3.21 *If* $S \rightarrow aSb \mid aAb$, $A \rightarrow bAa$, $A \rightarrow ba$ *is the context free grammar. Determine the context free language.*

AU : Dec.-11, Marks 6

Solution : The strings generated by this grammar are {a ba b, aa babb, aaa bbaa bbb, ...}

This denotes the language $L = \{a^n b^m a^m b^n | n, m \geq 1\}$

Example 3.3.22 *Find the context free langauge for the following grammars*

1) $S \rightarrow aSbS \mid bSaS \mid \varepsilon$

2) $S \rightarrow aSb \mid ab$

AU : May-12, Marks 10

Solution : 1) Let us derive some strings from given grammar.

S	S	aSbS
aSbS	bSaS	aaSbSbS
abSaSbS	baS	aabSbS
abaSbS	ba	aabbS
ababS		aabb
abab		

From these derivations, we can observe that the derived strings contain equal number of a's and b's. Thus this CFG denotes the language L containing equal number of a's and b's.

2) Refer example 3.3.2.

Example 3.3.23 *Construct a context free grammar for* $\{0^m 1^n | 1 \leq m \leq n\}$.

AU : Dec.-14, Marks 4

Solution : G = (V, T, P, S) where

V = {S, A}

T = {0,1}

P is a production rules set. It is given by -

$S \rightarrow 0S1 \mid 0A \mid 01$

$A \rightarrow 1A \mid 1$

3.4 Derivation Trees

Derivation trees is a graphical representation for the derivation of the given production rules for a given CFG. It is the simple way to show how the derivation can be done to obtain some string from given set of production rules. The derivation tree is also called parse tree.

Following are properties of any derivation tree -

1. The root node is always a node indicating start symbol.
2. The derivation is read from left to right.
3. The leaf nodes are always terminal nodes.
4. The interior nodes are always the non-terminal nodes.

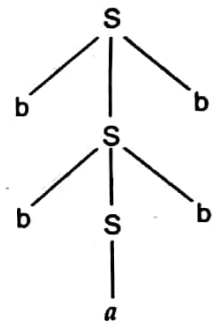
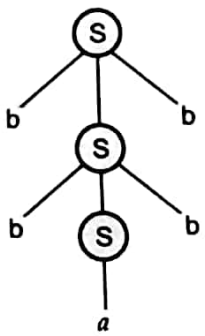


Fig. 3.4.1 Derivation tree

For example,

$S \rightarrow bSb \mid a \mid b$ is a production rule. The S is a start symbol.



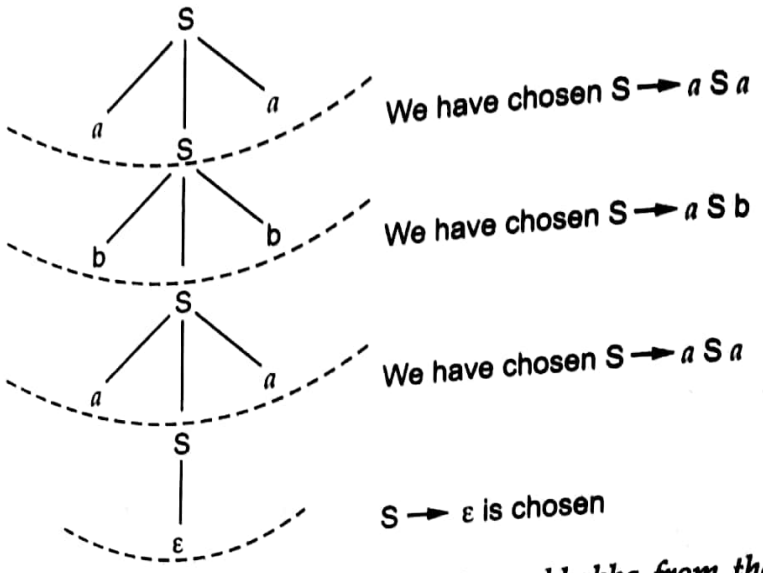
The above tree is a derivation tree drawn for deriving a string bbabb. By simply reading the leaf nodes we can obtain the desired string. The same tree can also be denoted by,

Examples for Understanding

Example 3.4.1 *Draw a derivation tree for the string abaaba for the CFG given by,*

G where $P = \{S \rightarrow aSa$
$S \rightarrow bSb$
$S \rightarrow a \mid b \mid \varepsilon\}$

Solution :



Example 3.4.2 *Construct the derivation tree for the string aabbabba from the CFG given by*

$S \rightarrow aB \mid bA$

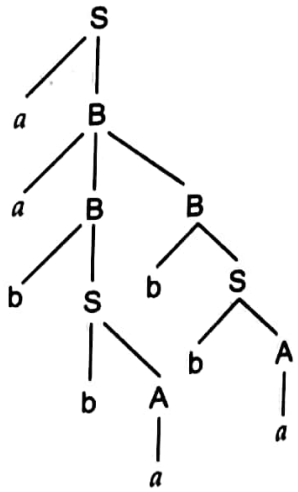
$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

Solution : To draw a tree we will first try to obtain derivation for the string *aabbabba*

S	
*a*B	S → *a*B
a [*a*BB]	S → *a*BB
aa [bS] B	B → bS
aab [bA] B	S → bA
aabb [*a*] B	A → *a*
aabba [bS]	B → bS
aabbab [bA]	S → bA
aabbabb [*a*]	A → *a*

Now let us draw a tree.



3.5 Relationship between Derivation and Derivation Trees

AU : Dec.-03, 04, 05, 10, 12, 14, May-04, 05, 09, 13, Marks 16

Theorem : Let G = (V, T, P, S) be a context free grammar. Then $S \overset{*}{\Rightarrow} a$ if and only if there is a derivation tree in grammar G which gives the string a.

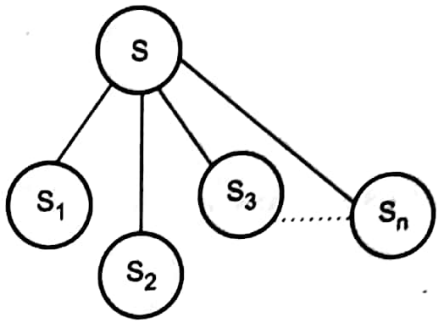


Fig. 3.5.1

Proof : For a non-terminal S there exists $S \overset{*}{\Rightarrow} w$ if and only if there is a derivation tree starting from root S and yielding w. To prove this we will use method of induction.

Basis of induction : Assume that there is only one interior node S. The derivation tree yielding $S_1, S_2, S_3 \ldots S_n$. From S is that means $S \overset{*}{\Rightarrow} S_1 S_2 \ldots S_n \overset{*}{\Rightarrow}$ a is input string.

Induction hypothesis : We assume that for K – 1 nodes the derivation tree can be drawn. We then can prove that for K vertices also we can have a derivation tree. That means the input string a can be derived as $S \rightarrow S_1 S_2 S_3 \ldots S_K$. There are two cases : either S_i may be a leaf variable or S_i may be an interior node yielding a. The S derives a by fewer number of K steps then $a \in S_1 S_2 S_3 S_4 \ldots S_K$.

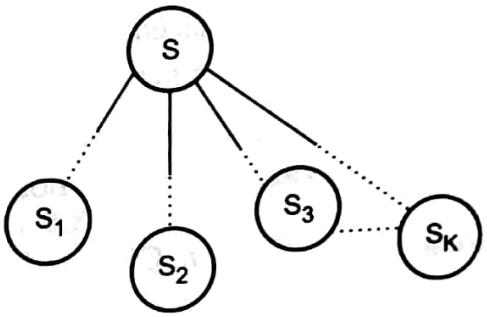


Fig. 3.5.2

If $a_i = S_i$ then S_i is leaf node (terminal) and if $S_i \overset{*}{\Rightarrow} a_i$ then S_i is an interior node. The tree for $S_1 S_2 \ldots S_k$ will be shown in Fig. 3.5.2

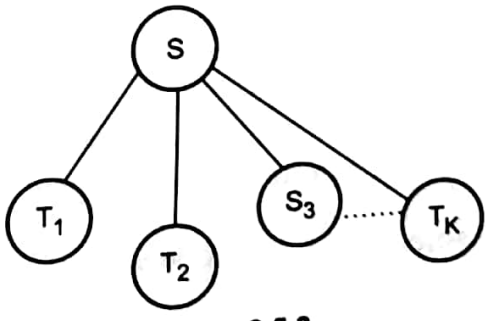


Fig. 3.5.3

We can also represent it as shown in Fig. 3.5.3.

This proves that $S \overset{*}{\Rightarrow} S_1, S_2, S_3 \ldots S_n \overset{*}{\Rightarrow}$ a can be obtained.

3.5.1 Leftmost Derivation and Rightmost Derivation

AU : May-09

As we know, derivation for any string means replacement of non-terminal by its appropriate definition. There may be a situation, in which there are many non-terminals.

Then which non-terminal should be replaced by its definition is sometimes confusing to decide.

Hence we normally apply two methods of deriving. The leftmost derivation is a derivation in which the leftmost non-terminal is replaced first from the sentential form.

The rightmost derivation is a derivation in which rightmost non-terminal is replaced first from the sentential form.

Let us see how it works.

For example

$S \rightarrow XYX$	$S \rightarrow XYX$
$S \rightarrow aYX$	$S \rightarrow XYa$
$S \rightarrow abX$	$S \rightarrow Xba$
$S \rightarrow aba$	$S \rightarrow aba$
Leftmost derivation	**Rightmost derivation**

Note that we have replaced first X from left to right in leftmost derivation and XYX the last X i.e. the rightmost symbol.

Actually, we may use leftmost derivation or rightmost derivation we get the same string. The type of derivation does not affect on getting of a string.

Theorem : Let G be a CFG and let $A \overset{*}{\Rightarrow} w$ is in G. Then show that there is a leftmost derivation of w.

AU : May-04, Marks 6

Proof : We will use the method of induction to prove this theorem.

Basis : We will assume height of derivation tree is 1. Then the smallest parse tree as shown below will be generated.

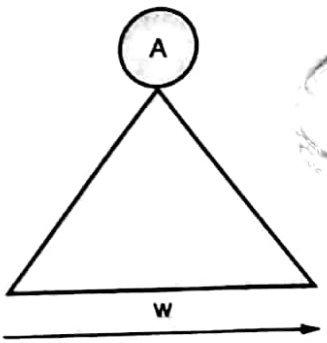


Fig. 3.5.4

Then surely $A \underset{lm}{\overset{*}{\Rightarrow}} w$ is one step.

Induction : We assume that height = n, where n > 1. Then the derivation tree for $A \overset{*}{\Rightarrow} w$ is as shown below.

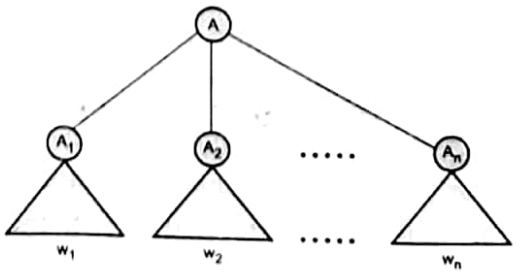


Fig. 3.5.5

There are two cases.

i) If A_i is terminal then we can define w_i is a string for A_i.

ii) If A_i is non-terminal (variable) then there must be some leftmost derivation $A_i \overset{*}{\Rightarrow} w_i$.

The $w_i = w_1\ w_2\ w_3 \ldots w_n$ and to construct leftmost derivation for n nodes.

$$A \overset{*}{\underset{lm}{\Rightarrow}} A_1\ A_2\ A_3 \ldots A_n \text{ for each } i = 1, 2, 3, \ldots n \text{ in order.}$$

$$\therefore \quad A \overset{*}{\underset{lm}{\Rightarrow}} w_1\ w_2 \ldots w_{i-1}\ A_i\ A_i + 1\ A_{i+2} \ldots A_n$$

Again if i) A_i is terminal then

$$A \overset{*}{\underset{lm}{\Rightarrow}} w_1\ w_2 \ldots w_i\ A_{i+1}\ A_{i+2} \ldots A_n$$

And if ii) A_i is a non-terminal then, we continue the leftmost derivation of w_i from A_i such that $A_i \underset{lm}{\Rightarrow} \alpha_1 \underset{lm}{\Rightarrow} \alpha_2 \ldots \underset{lm}{\Rightarrow} w_i$

We proceed

$$A \Rightarrow w_1\ w_2 \ldots w_{i-1}\ A_i\ A_{i+1} \ldots A_n \underset{lm}{\Rightarrow} w_1\ w_2 \ldots \alpha_1\ A_i\ A_{i+1} \ldots A_n \underset{lm}{\Rightarrow} w_1\ w_2$$

$$\ldots \alpha_2\ A_i\ A_{i+1} \ldots A_n \underset{lm}{\Rightarrow} w_1\ w_2\ w_3 \ldots w_i\ A_{i+1}\ A_{i+2} \ldots A_n$$

This means

$$A \overset{*}{\underset{lm}{\Rightarrow}} w_1\ w_2 \ldots w_i\ A_{i+1}\ A_{i+2} \ldots A_n$$

i.e. $A \overset{*}{\underset{lm}{\Rightarrow}} w$

This proves that there is a leftmost derivation of w.

Examples for Understanding

Example 3.5.1 *Let, G be the grammar*

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

For the string baaabbabba. Find leftmost derivation, rightmost derivation and Parse tree.

AU : Dec.-10, Marks 9

Solution :

Leftmost derivation	Rightmost derivation
S	S
bA	bA
baS	baS
baaB	baaB
baaaBB	baaaBB
baaabSB	baaaBbS
baaabbAB	baaaBbbA
baaabbaB	baaaBbba
baaabbabS	baaabSbba
baaabbabbA	baaabbAbba
baaabbabba	baaabbabba

Parse tree :

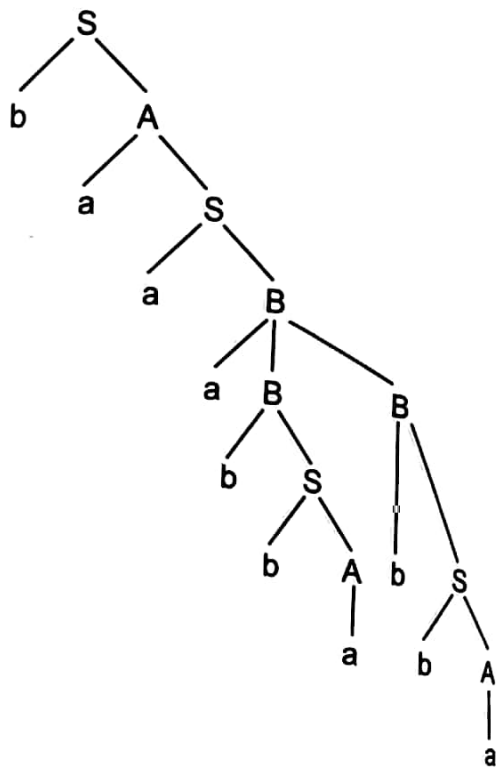


Fig. 3.5.6

Example 3.5.2 *Derive the string 1000111 for leftmost and rightmost derivation using CFG.*

$G \equiv (V, T, P, S)$ where

$V = \{S, T\}$

$T = \{0, 1\}$

$P = \{ S \rightarrow T00T$

$T \rightarrow 0T \mid 1T \mid \varepsilon \ \}$

Solution : The leftmost derivation can be

S

T00T $\quad S \rightarrow T00T$

1T00T $\quad T \rightarrow 1T$

10T00T $\quad T \rightarrow 0T$

10 ε 00T $\quad T \rightarrow \varepsilon$

1000T

10001T $\quad T \rightarrow 1T$

100011T $\quad T \rightarrow 1T$

1000111T $\quad T \rightarrow 1T$

1000111 ε $T \rightarrow \varepsilon$

1000111

Right most derivation -

S

T00T

T001T $T \rightarrow 1T$

T0011T $T \rightarrow 1T$

T00111T $T \rightarrow 1T$

T00111 ε $T \rightarrow \varepsilon$

T00111

1T00111 $T \rightarrow 1T$

10T00111 $T \rightarrow 0T$

10 ε 00111 $T \rightarrow \varepsilon$

1000111

Example 3.5.3 *Consider the following grammar for list structures :*

$S \rightarrow a \mid \wedge \mid (T)$

$T \rightarrow T, S \mid S$

Find left most derivation, rightmost derivation and Parse tree for (((a, a), ^ (a)), a)

AU : Dec.-12, Marks 10

Solution : We will assume the string as

((a, a), (^, a)), a)

Leftmost Derivation	Rightmost Derivation
S	S
(T)	(T)
(T, S)	(T, S)
(S, S)	(T, a)
((T), S)	(S, a)
((T, S), S)	((T) a)
((S, S), S)	((T, S), a)
(((T), S), S)	((T, (T)), a)
(((T, S), S), S)	((T, (T, S)), a)
(((S, S),S), S)	((T, (S, S)), a)
(((a, S), S), S)	((T, (S, a)), a)

(((a, a), S), S)	((T, (^, a)), a)
(((a, a),(T)), S	((S, (^, a)), a)
(((a, a), (T, S)), S)	(((T), (^, a)), a)
(((a, a), (S, S)), S)	(((T, S), (^, a)), a)
(((a, a), (^, S)), S)	(((T, a), (^, a)), a)
(((a, a), (^, a)), S)	(((S, a), (^, a)), a)
(((a, a), (^, a)), a)	(((a, a), (^, a)), a)

Parse Tree

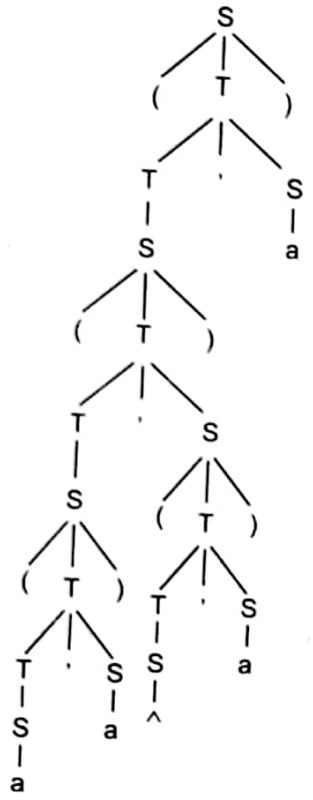


Fig. 3.5.7

Example 3.5.4 *Consider the following productions*

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

For the string aaabbabbba, Find

1) Leftmost derivation

2) Rightmost derivation

3) Parse tree

AU : May-13, Marks 8

Solution : For leftmost derivation and parse tree refer answer of Q.10 on page 3-67.

Rightmost Derivation :

S	rule S → aB
a B	rule B → aBB
a a B B	rule B → bS
a a B b S	rule S → bA
a a B b b A	rule A → a
a a B b b a	rule B → aBB
a a a B B b b a	rule B → b
a a a B b b b a	rule B → bS
a a a b S b b b a	rule S → bA
a a a b b A b b b a	rule A → a
a a a b b a b b b a	

Example 3.5.5 *Write a grammar G to recognize all prefix expressions involving all binary arithmetic operators. Construct a parse tree for the sentense '_*+abc / de' using G ?*

AU : Dec.-14, Marks 16

Solution : G = (V, T, P, S) where

V = {S, A, B, C, D, E}

T = {+,–,*,/, a, b, c, d, e}

S is a start symbol.

The production rules are

S → A | B | C | D | E

A → + SS

B → – SS

C → * SS

D → /SS

E → a | b | c | d

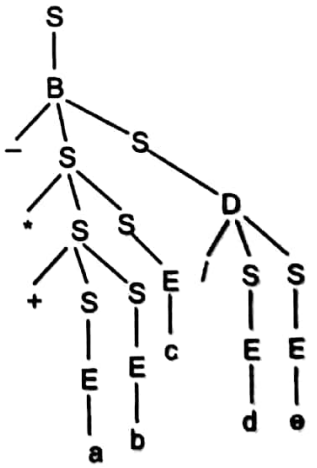


Fig. 3.5.8 Parse tree

Review Question

1. *Let G = (V, T, P, S) be a context free grammar. Then prove that $S \overset{*}{\Rightarrow} \alpha$ if and only if there is a derivation tree in grammar G with yield α.*

AU : Dec.-03, Marks 16; Dec.-04, Marks 6; May-05, Marks 12; Dec.-05, Marks 10

3.6 Ambiguity

AU : May-04, 07, 09, 10, 12, 14, Dec.-06, 10, 13, 14, Marks 16

The grammar can be derived in either leftmost derivation or rightmost derivation. One can draw a derivation tree called as parse tree or syntax tree based on these derivations. The parse tree has to be unique even though the derivation is leftmost or rightmost.

But if there exists more than one parse trees for a given grammar, that means there could be more than one leftmost or rightmost derivation possible and then that grammar is said to be ambiguous grammar.

Example for Understanding

Example 3.6.1 *The CFG given by* $G = (V, T, P, S)$

where $V = \{E\}$

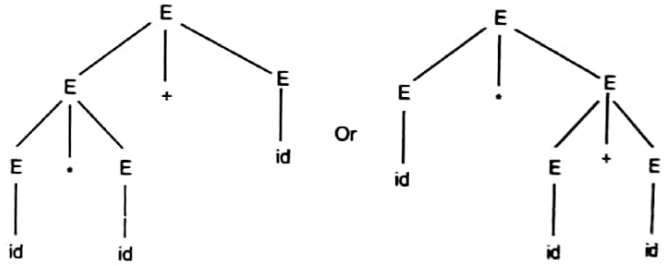
$T = \{id\}$

$P = \{E \rightarrow E + E,\ E \rightarrow E * E$

$E \rightarrow id\ \}$

$S = \{E\}$. *Is the grammar ambiguous ?* **AU : Dec.-10, Marks 2, May-12, Marks 6**

Solution : Now if the string is **id * id + id** then we can draw the two different parse trees indicating our id * id + id.



Thus the above grammar is an ambiguous grammar.

Solved Examples

Example 3.6.2 *Consider the grammar,*

i) $S \rightarrow i\ C\ t\ s$

$S \rightarrow i\ C\ t\ S\ e\ S$

$S \rightarrow a$

$C \rightarrow b$

where **i, t** *and* **e** *stand for* **if, then** *and* **else** *and* C *and* S *for "conditional" and "statement" respectively.*

1) *Construct leftmost derivation for the sentence* $W =$ ***libtibtaea.***
2) *Show the corresponding Parse tree for the above sentence.*
3) *Is the above grammar ambiguous? If so prove it.*
4) *Remove ambiguity if any and prove that both the grammar produces the same language.*

AU : May-10, Marks 16

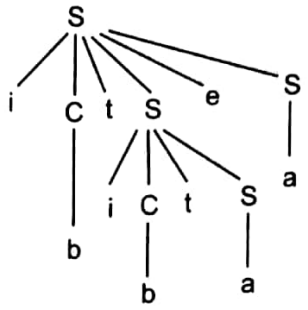
Solution :

1)

S
iCtSeS
ibtSeS
ibtiCtSeS
ibtibtSeS
ibtibtaeS
ibtibtaea

(a)

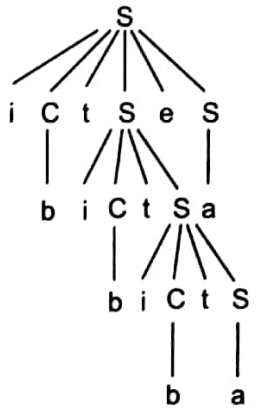
2)



(b)

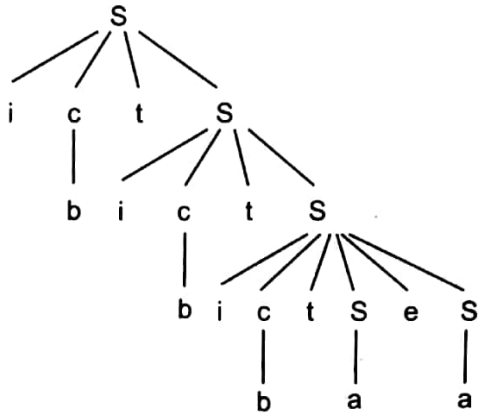
Fig. 3.6.1

3) Yes the given grammar is an ambiguous grammar because it produces two different parse trees for deriving the same input string. for example - For deriving the input ibtibtibtaea we can draw two different parse trees.



Parse tree 1

(c)



Parse tree 2

(d)

Fig. 3.6.1

4) For making the given grammar unambiguous we match each **else** with closest previous **unmatched then**. Hence on removing ambiguity we will get following rules -

$S \rightarrow M \mid U$

$M \rightarrow iCtMeM \mid a$

$U \rightarrow iCtS \mid iCtMeU$

$C \rightarrow b$

Here we consider M and U non-terminals for matched and unmatched statements.

Example 3.6.3 *If G is the grammar $S \rightarrow SbS \mid a$. Show that G is ambiguous.*

AU : May-04, 14, Dec.-14, Marks 6

Solution : For the derivation of same string if there exists more than one parse tree then that grammar is supposed to be an ambiguous grammar.

Let w = *ababa* be the string such that w ∈ G then, for production rules

S → SbS

S → *a*

The derivation trees are

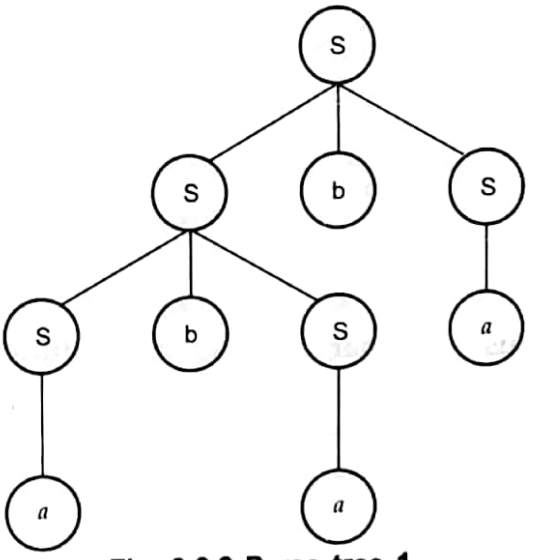


Fig. 3.6.2 Parse tree 1

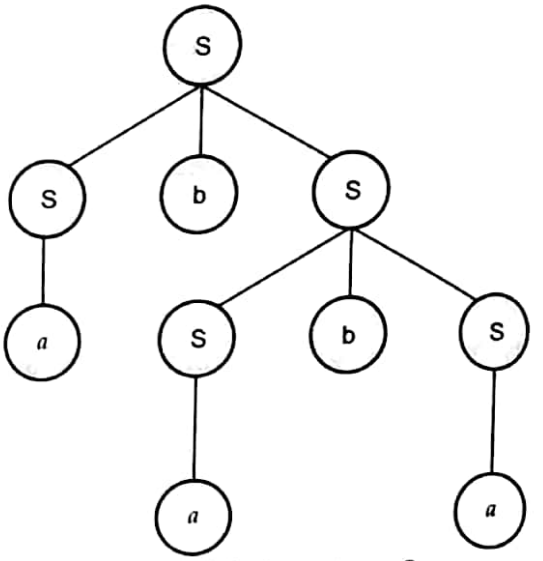


Fig. 3.6.3 Parse tree 2

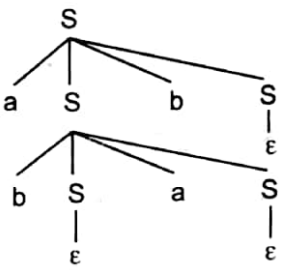
Example 3.6.4 *Show that the grammar S → a S b S | b S a S | ε is ambiguous and what is the language generated by this grammar?* **AU : Dec.-06, Marks 6**

Solution : To show that given language is ambiguous we have to draw parse trees. If there exists more than one parse trees for some string then that grammar is said to be ambiguous grammar.

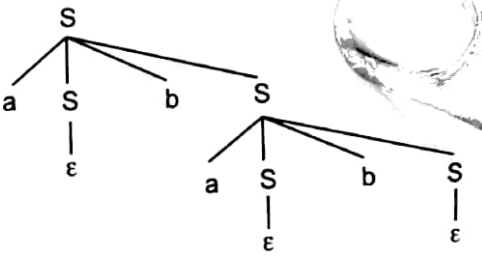
Consider the string "abab", we can draw -

The given grammar is ambiguous.

The language generates all the strings having even length with equal number of a's and b's.



(a) Parse tree 1



(b) Parse tree 2

Fig. 3.6.4

Example 3.6.5 *Show that the grammar S → a | abSb | aAb, A → bS | aAAb is ambiguous.* **AU : May-07, Marks 6**

Solution : A grammar is said to be an ambiguous grammar if it has more than one derivation trees for deriving a string that belongs to this language. Let us denote this grammar by G and there is a string 'abababb' that belongs to L(G). Following two different trees derive the same string.

Hence the grammar is an ambiguous.

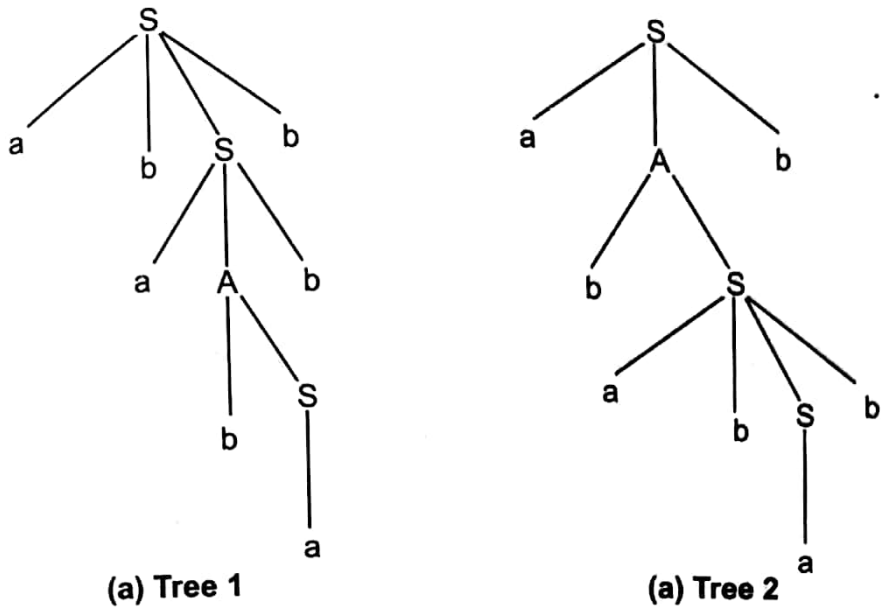


Fig. 3.6.5

Example 3.6.6 *Consider the following grammar G with productions*

$S \rightarrow 0B/1A \quad A \rightarrow 0/0S/1AA \quad B \rightarrow 1/1S/0BB.$

for the sentence $\omega = 00110101$

i) Construct a leftmost derivation and right most derivation.

ii) Show the corresponding parse tree for the above sentence.

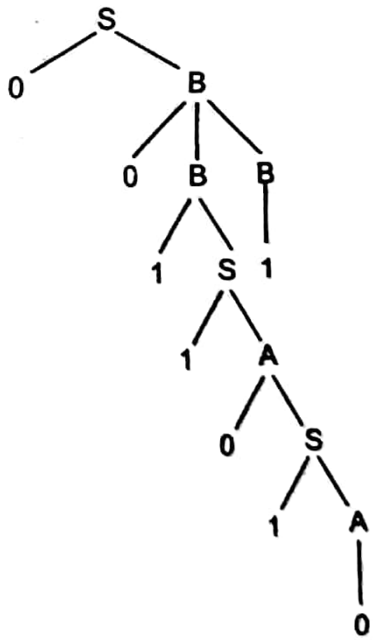
iii) Is the above grammar ambiguous ? If so, prove it.

Solution :

i)

Leftmost derivation	Rightmost derivation
S → 0B	S → 0B
00BB	00BB
001SB	00B1
0011AB	001S1
00110SB	0011A1
001101AB	00110S1
0011010B	001101A1
00110101	00110101

ii)



iii) The given grammar is ambiguous.

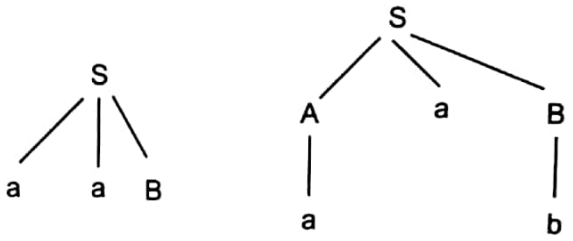
Example 3.6.7 *Show that the following grammars are ambiguous.*

{S → aSbS | bSaS | λ} and

{S → AB | aaB, A → a | Aa, B → b}

Dec.-13, Marks 6

Solution : i) Refer example 3.6.4.



(a) Parse tree 1 **(b) Parse tree 2**

Fig. 3.6.6

ii) We will generate the string **aab**.

As there are two different parse trees that can be drawn for the string **aab**. Hence the given grammar is ambiguous.

University Question

1. *Define ambiguity, left most derivation and right most derivation with an example.*

AU : May-09, Marks 6

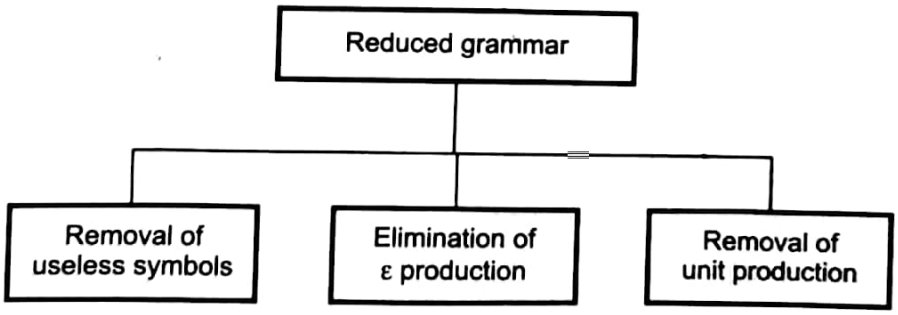
3.7 Simplification of CFG

AU : Dec.-07, May-11, Marks 4

As we have seen various languages can effectively be represented by context tree grammar. All the grammars are not always optimized. That means grammar may consists of some extra symbols (non-terminals). Having extra symbols unnecessary increases the length of grammar. Simplification of grammar means reduction of grammar by removing useless symbols. The properties of reduced grammar are given below.

1. Each variable (i.e. non-terminal) and each terminal of G appears in the derivation of some word in L.
2. There should not be any production as $X \rightarrow Y$ where X and Y are non-terminals.
3. If ε is not in the language L then there need not be the production $X \rightarrow \varepsilon$.

We see the reduction of grammar as below.



Let us study the reduction process in detail.

4 Pushdown Automata

Syllabus

Pushdown automata - Definitions - Moves - Instantaneous descriptions - Deterministic pushdown automata - Equivalence of pushdown automata and CFL - Pumping lemma for CFL - Problems based on pumping lemma.

Contents

4.1 Definition

AU : May-10, Marks 2

The pushdown automata will have **input tape, finite control and stack.**

The input tape is divided in many cells. At each cell only one input symbol is placed thus certain input string is placed on tape. The finite control has some pointer which point the current symbol which is to be read. At the end of input $ or Δ (blank) b (blank) symbol is placed which indicates end of input.

The stack is such a structure in which you can push and remove the items from one end only. For example if you place some coins one above the other, then a stack of coins is formed. While removing a coin we can only remove the coin which is at the top. Similarly while adding more coins to the stack we can place a new coin only on the topmost coin. Thus the stack of coin shows clear cut use of only one end of the stack.

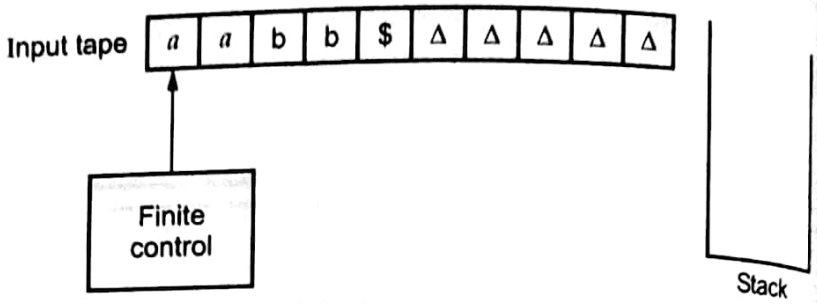


Fig. 4.1.1 Pushdown automata

In the pushdown automata, we are using the stack for storing the items temporarily. Inserting the symbol onto the stack is called push operation and removing a symbol from stack is called pop operation.

Let us have a formal definition of pushdown automata (PDA).

The PDA can be defined as a collection of seven components.

1. The finite set of states Q.
2. The input set Σ.
3. Γ is a stack alphabet.
4. q_0 is initial stage, $q_0 \in Q$.
5. Z_0 is a start symbol which is in Γ.
6. Set of final states $F \subseteq Q$.
7. δ is mapping function used for moving from current state to next state.

Following symbols are used while drawing the pushdown automata.

As we have discussed earlier PDA is more powerful than FA. Any language which can be accepted by FA can also be accepted by PDA. Not even this, PDA accepts a class of languages which even can not be accepted by FA. Thus PDA is much more superior to FA. Let us see how it works.

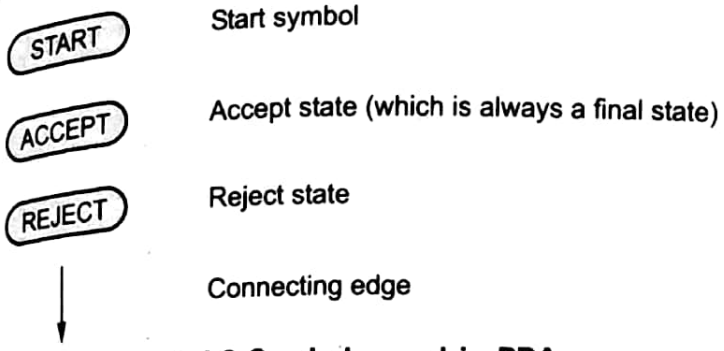


Fig. 4.1.2 Symbols used in PDA

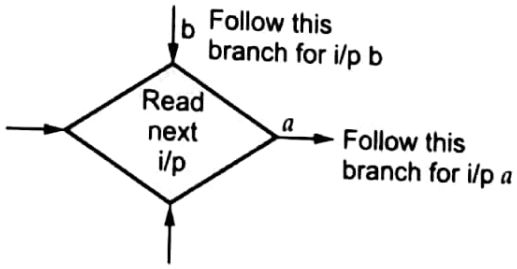


Fig. 4.1.3 Read symbol

Example 4.1.1 *Design pushdown automata which accepts only odd number of a's over* $\Sigma = \{a, b\}$.

Solution : This problem is similar to the one which we have solved for drawing FA. We will draw a finite automata for the same as -

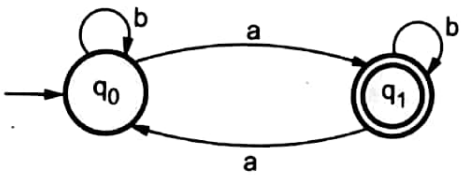


Fig. 4.1.4

We will first make some observations -

i) If initially 'b' is read then the self loop is to be applied.

ii) If we read 'a' initially then the read state gets change [in FA state change from q_0 to q_1 occurs] and if again a is read then we return back to previous state.

iii) After a if we read a then a self loop is applied.

iv) We basically read odd number of a's and any number of b's.

With these observations the PDA can be constructed as -

Note that this PDA is very much resembling to the finite automata drawn. We will simulate this PDA for some string.

Consider input string "baaab" is placed on input tape.

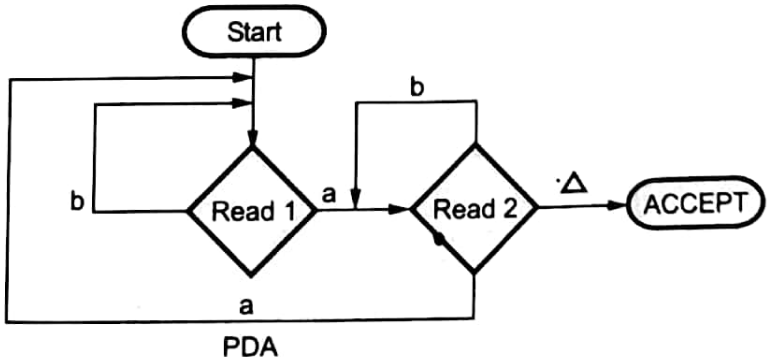
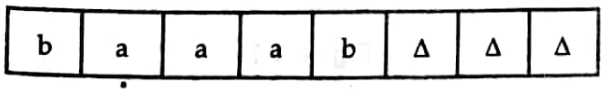


Fig. 4.1.5 PDA



We will read only one character at a time from left to right. As soon as we read Δ we should reach to **ACCEPT** state of PDA.

```
b a a a b Δ        Read 1
↑
b a a a b Δ        Read 2
 ↑
b a a a b Δ        Read 1
   ↑
b a a a b Δ        Read 2
     ↑
b a a a b Δ        Read 2
       ↑
b a a a b Δ        ACCEPT
         ↑
```

This shows that only odd number of a's are allowed and there is no restriction on number of b's.

Example 4.1.2 *Design PDA for the language L = {001}.*

Solution : The simple FA for this language is

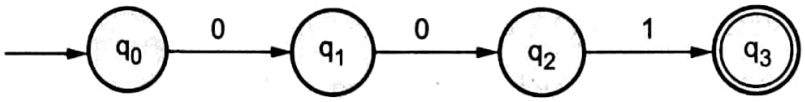


Fig. 4.1.6

where q_0 is a start state and q_3 is a accept state. We can draw an equivalent PDA as

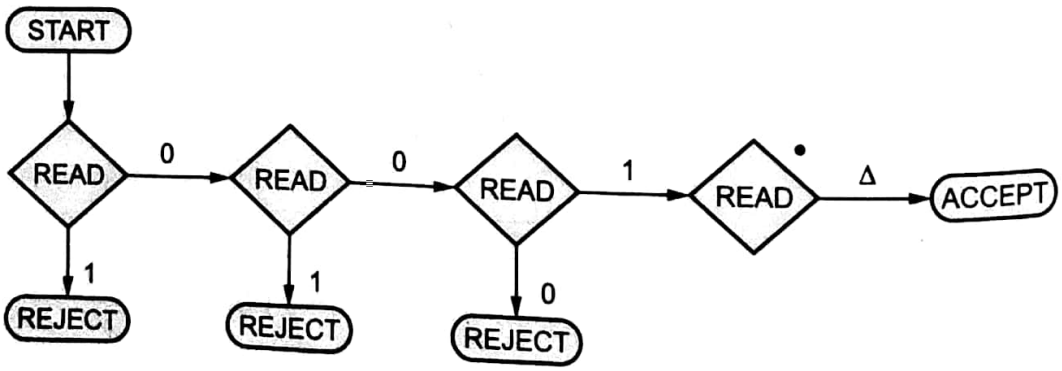


Fig. 4.1.7

You can note that first read node resembles state q_0, second read statement resembles q_1 state and third read statement resembles q_2 state, and so on. After final read statement it will lead to accept state.

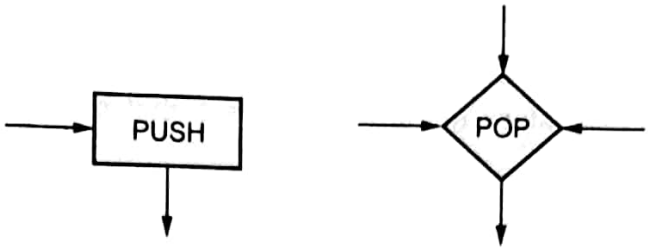


Fig. 4.1.8

From next problem onwards we will introduce two more new symbols.

Example 4.1.3 *Design a PDA for the language* $L = \{a^n b^n\}$ *where* $n \geq 1$.

Solution : This is a typical example which can be solved by the PDA. Note that we can not draw finite automata for this problem because, here the condition is that what many number of *a*'s are occuring those many b's should be after *a*'s. That means if n = 5 then the string will be *aaaaa*bbbbb. If we try to design FA for this we are unable to keep track of how many *a*'s or b's have occured, since there is no memory in FA.

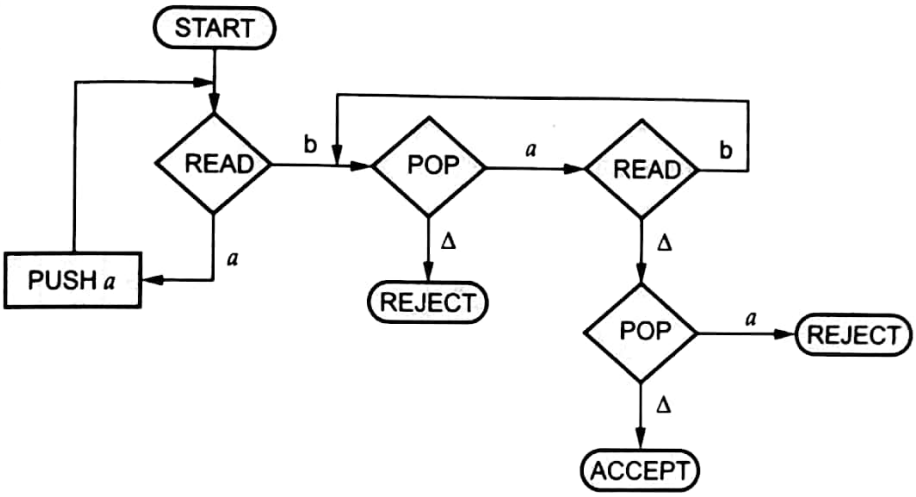


Fig. 4.1.9

Let us draw PDA for the same using memory i.e. stack.

The simple logic we have applied here is that we are starting with start symbol. At the first read statement if we read *a*, we push that *a* onto the stack. This will be repeated by applying a self loop to this read statement. By this all the *a*'s will be pushed onto the stack. Now when we will read one b we will pop one *a*. Thus, this process will be repeated till the end of input. The end marker is Δ. After reading Δ from input tape, we pop the contents from the stack. If we pop a that means number of *a*'s are more than total number of b's. So we go to a reject state. Let us simulate this machine for some valid string.

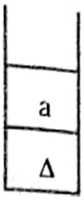
a	*a*	b	b	Δ	Δ	Δ

Input tape for $a^n b^n$, now n = 2.

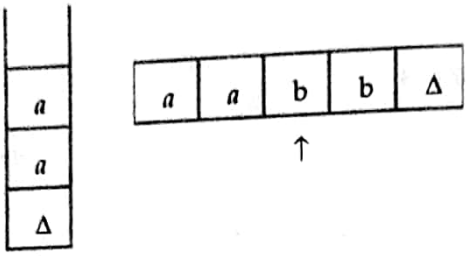
Step 1 : We will start with start symbol.

Step 2 : We will reach to a read state, we will read first *a* and push it onto the stack.

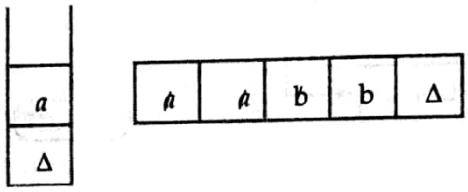
The stack initially, contains Δ that means it is empty.



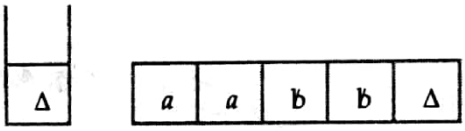
Step 3 : Read next symbol from input tape. It is *a* and push it onto the stack.



Step 4 : Read b from the tape and pop from the stack if the popped symbol is Δ it will go in reject states. That means number of *a*'s are lesser than b's. So after reading first b, we pop one *a*.



After reading second b we pop one more *a*.



Note that both tape and stack both are empty.

Example 4.1.4 *Design PDA that checks the well formedness of parenthesis.*

Solution : The well formedness of parenthesis means that the input should start with '(' opening parenthesis and it should be followed by ')' closing parenthesis. The number of '(' and ')' should be equal. The input should not start with ')'. For example (( )) or ( ) ( ) ( ) or (( )( )( )) . Thus we will design PDA which will check a valid string.

Let us simulate this PDA for the input

Hence we reach to accept state.

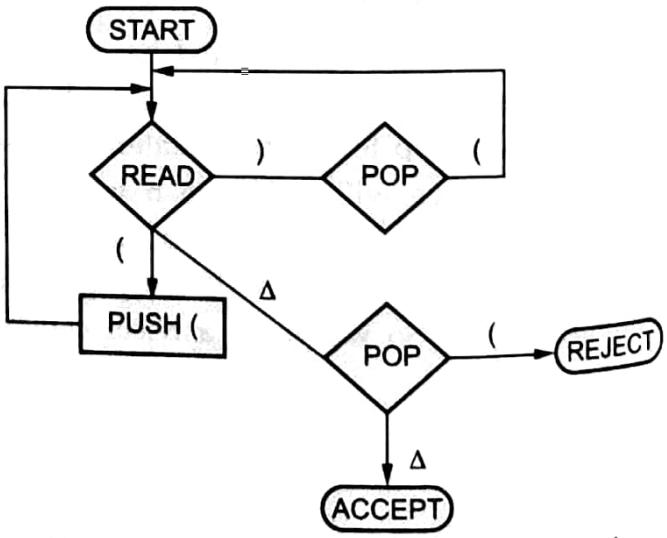


Fig. 4.1.10 Well formedness of parenthesis

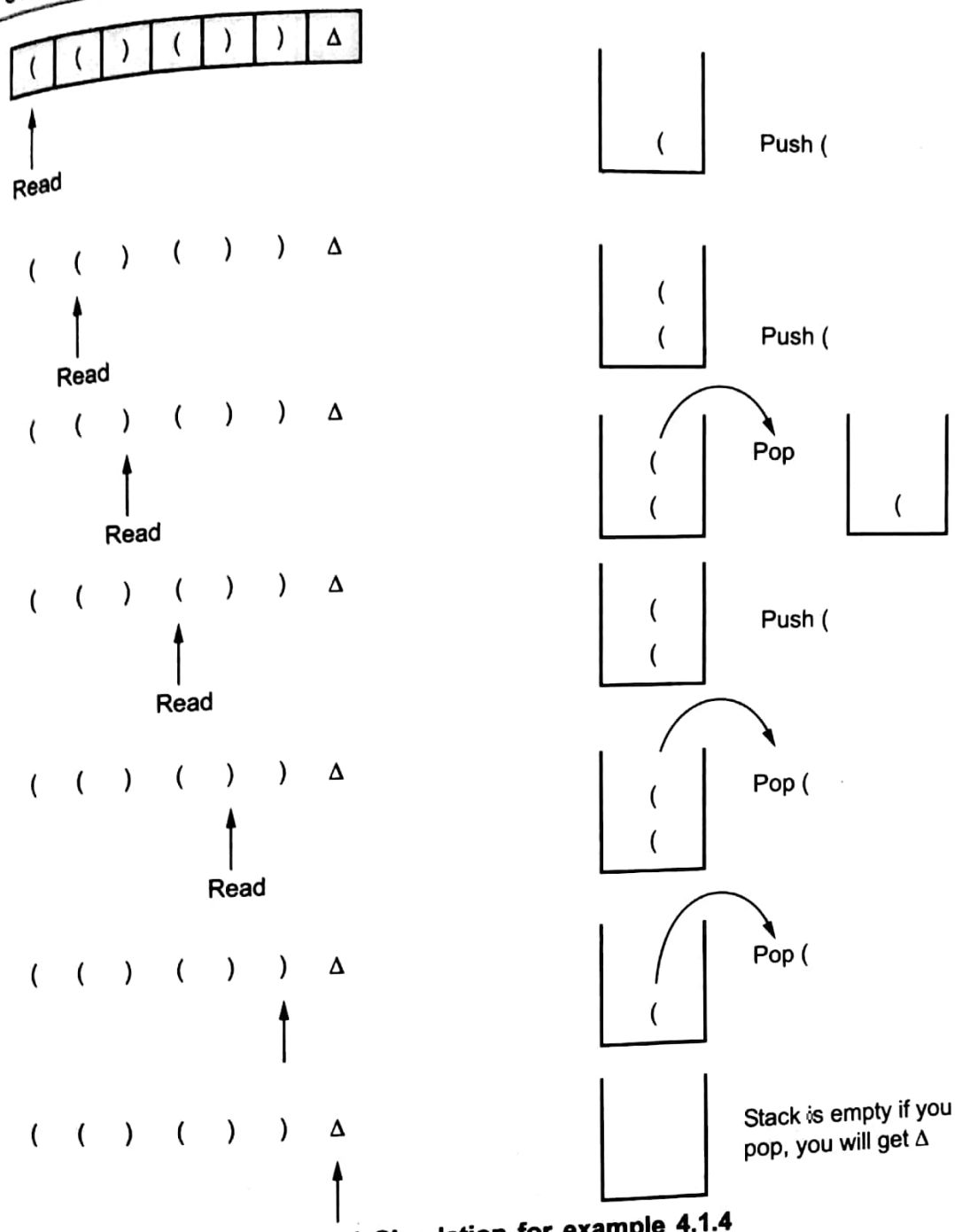


Fig. 4.1.11 Simulation for example 4.1.4

4.2 Instantaneous Description

AU : Dec.-07, 09, 10, 12, 13, May-07, 11, 12, 13, 14, Marks 8

As we have seen a pictorial model of PDA for some typical problems like $a^n b^n$ and well formedness of parenthesis. Also we have experienced one thing that for representing certain problems FA is not sufficient, we need PDA wherein there is an effective use of stack as a memory. We can describe this behaviour of PDA by means of instantaneous description. It is given by ID. The moves of ID's are as shown below.

From above Fig. 4.2.1, if we are reading the current symbol a_2, at current state S_1 and current stack symbol Z_1 then after a move we will reach to state S_2 and there will be some new symbol on the top of the stack. This description can be represented as -

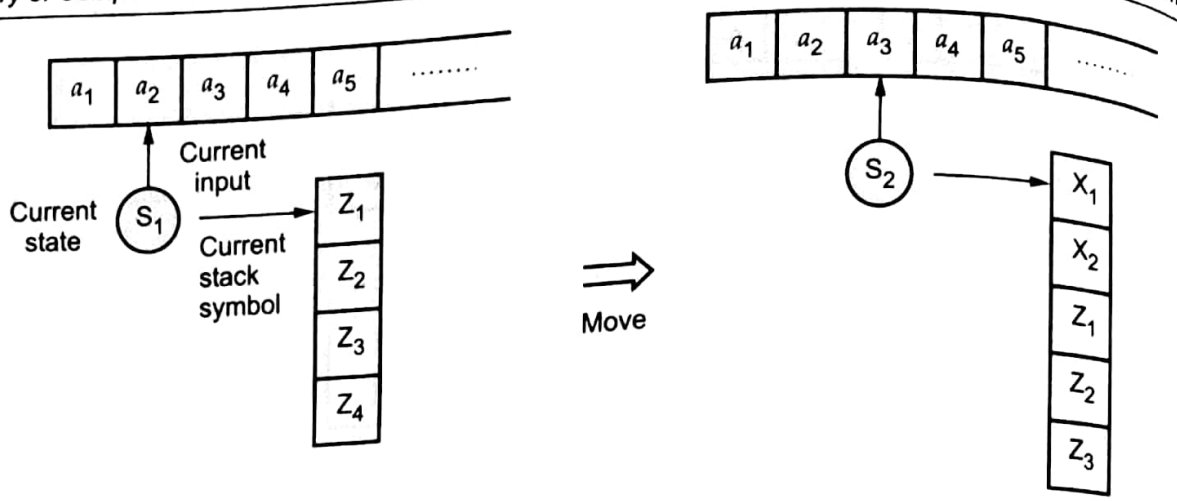


Fig. 4.2.1

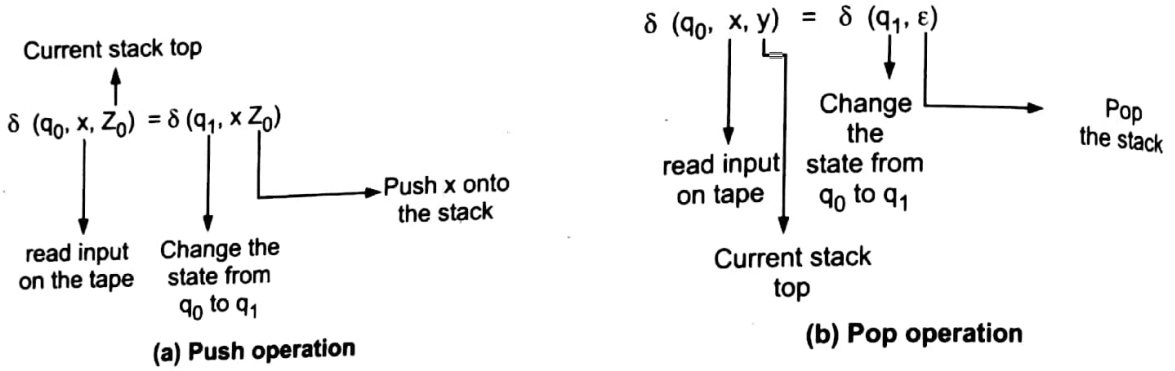


Fig. 4.2.2 Instantaneous description

Now let us solve some example based on it.

Example for Understanding

Example 4.2.1 *Design a PDA for accepting a language* $\{L = a^n b^n \mid n \geq 1\}$.

AU : May-14, Marks 10

Solution : This is a language in which equal number of a's are followed by equal number of b's. The logic for this PDA can be applied as : first we will push all a's onto the stack. Then on reading every single b each a is popped from the stack. If we read all b and remove all a's and if we get stack empty then that string will be accepted. The instantaneous description can be given as -

$$\delta\ (q_0, a, Z_0) = (q_0, aZ_0)$$

$$\delta\ (q_0, a, a) = (q_0, aa)$$

$$\delta\ (q_0, b, a) = (q_1, \varepsilon)$$

$$\delta\ (q_1, b, a) = (q_1, \varepsilon)$$

$$\delta\ (q_1, \varepsilon, Z_0) = (q_2, \varepsilon)$$

where q_0 is a start state and q_2 is accept state.

We will simulate this PDA for following string -

$(q_0, aaabbb, Z_0) \vdash (q_0, aabbb, aZ_0)$

$\vdash (q_0, abbb, aaZ_0)$

$\vdash (q_0, bbb, aaaZ_0)$

$\vdash (q_1, bb, aaZ_0)$

$\vdash (q_1, b, aZ_0)$

$\vdash (q_1, \varepsilon, Z_0)$

$\vdash (q_2, \varepsilon)$

ACCEPT state.

Solved Examples

Example 4.2.2 *Construct PDA for the language* $L = \{a^n b^{2n} \mid n \geq 1\}$ **AU : Dec.-07, Marks 8**

Solution : In this language n number of a's should be followed by 2n number of b's. Hence we will apply a very simple logic and that is if we read single 'a' we will push two a's onto the stack. As soon as we read 'b' then for every single 'b' only one 'a' should get popped from the stack. This basically maintains the $a^n b^{2n}$ count and sequence. The ID can be constructed as follows -

$$\delta(q_0, a, Z_0) = (q_0, a\, a\, Z_0)$$

$$\delta(q_0, a, a) = (q_0, a\, a\, a)$$

Now when we read b we will change the state from q_0 to q_1 and start popping corresponding 'a'. Hence ,

$$\delta(q_0, b, a) = (q_1, \varepsilon)$$

Thus this process of popping will be repeated unless all the symbols are read. Note that popping action occurs in state q_1 only.

$$\therefore \quad \delta(q_1, b, a) = (q_1, \varepsilon)$$

After reading all b's all the corresponding a's should get popped. Hence when we read ε as input symbol there should be nothing in the stack. Hence the move will be -

$$\delta(q_1, \varepsilon, Z_0) = (q_2, \varepsilon)$$

where the PDA $P = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \{q_2\})$

We can summarize the ID as

$$\delta(q_0, a, Z_0) = (q_0, aaZ_0)$$

$\delta(q_0, a, a) = (q_0, aaa)$

$\delta(q_0, b, a) = (q_1, \varepsilon)$

$\delta(q_1, b, a) = (q_1, \varepsilon)$

$\delta(q_1, \varepsilon, Z_0) = (q_2, \varepsilon)$

Let us simulate this PDA for some input string "aaabbbbbb"

$\delta(q_0, aaabbbbbb, Z_0) \vdash (q_0, aabbbbbb, aaZ_0)$

$\vdash (q_0, abbbbbb, aaaaZ_0)$

$\vdash (q_0, bbbbbb, aaaaaaZ_0)$

$\vdash (q_1, bbbbb, aaaaaZ_0)$

$\vdash (q_1, bbbb, aaaaZ_0)$

$\vdash (q_1, bbb, aaaZ_0)$

$\vdash (q_1, bb, aaZ_0)$

$\vdash (q_1, b, aZ_0)$

$\vdash (q_1, \varepsilon, Z_0)$

$\vdash (q_2, \varepsilon)$

Final state or ACCEPT state.

Thus input gets accepted by using the constructed PDA..

AU : Dec.-09, Marks 8; Dec.-10, Marks 7

Example 4.2.3 *For a given PDA* $M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_2\})$, *the mapping function* δ *is given as*

$R_1 : \delta(q_0, a, Z_0) = \delta(q_0, aZ_0)$

$R_2 : \delta(q_0, b, Z_0) = \delta(q_0, bZ_0)$

$R_3 : \delta(q_0, a, a) = \delta(q_0, aa)$

$R_4 : \delta(q_0, b, a) = \delta(q_0, ba)$.

$R_5 : \delta(q_0, a, b) = \delta(q_0, ab)$

$R_6 : \delta(q_0, b, b) = \delta(q_0, bb)$

$R_7 : \delta(q_0, c, Z_0) = \delta(q_1, Z_0)$

$R_8 : \delta(q_0, c, a) = \delta(q_1, a)$

$R_9 : \delta(q_0, c, b) = \delta(q_1, b)$ $R_{10} : \delta(q_1, a, a) = \delta(q_1, \varepsilon)$

$R_{11} : \delta(q_1, b, b) = \delta(q_1, \varepsilon)$

$R_{12} : \delta(q_1, \varepsilon, Z_0) = \delta(q_2, Z_0)$

Simulate for the input i) bbacabb

Solution : We will first draw the transition graph for given δ transitions.

Now we will simulate this PDA for the input bbacabb.

From initial ID, we will try to match rules R_1 to R_{12}.

$(q_0, \underline{b}bacabb, \underline{Z}_0) \vdash (q_0, \underline{b}acabb, \underline{b}Z_0)$

$\vdash (q_0, \underline{a}cabb, \underline{b}bZ_0)$

$\vdash (q_0, cabb, abbZ_0)$

$\vdash (q_1, \underline{a}bb, \underline{a}\,bbZ_0)$

$\vdash (q_1, \underline{b}b, \underline{b}bZ_0)$

$\vdash (q_1, b, bZ_0)$

$\vdash (q_1, \varepsilon, Z_0)$

$\vdash (q_2, Z_0)$

which is ACCEPT state.

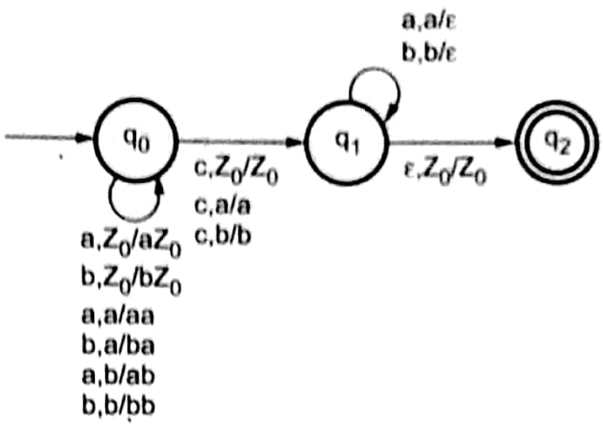


Fig. 4.2.3

Thus the input bbacabb is accepted by given PDA.

This is a language $L(M) = \{wCw^R \mid w \in (a + b)^*\}$ where w^R is reverse of w.

Example 4.2.4 *Design PDA to accept the language* **AU : May-07, Marks 8**

$L = \{w/w \in (a + b)^* \text{ and } n_a(w) = n_b(w)\}$

Solution : This is a language of equal number of a's and equal number of b's. Initially when stack is empty then whatever we read either 'a' or 'b' we will simply push it onto the stack. Now if we read 'a' and at the top of the stack if 'b' is present then it will be erased by ε. Similarly if we read 'b' and top of the stack contains 'a' then erase it by ε. The instantaneous description for this language is as given below -

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$

$\delta(q_0, b, Z_0) = (q_0, bZ_0)$

$\delta(q_0, a, a) = (q_0, aa)$

$\delta(q_0, b, b) = (q_0, bb)$

$\delta(q_0, a, b) = (q_0, \varepsilon)$

$\delta(q_0, b, a) = (q_0, \varepsilon)$

$\delta(q_0, \varepsilon, Z_0) = (q_1, Z_0)$

Where q_0 is a start state and q_1 is a final state. When we reach to this state with ε input and having an empty stack. We move to accept/final state. Let us simulate this for a string 'aababb'.

$\delta(q_0, aababb, Z_0) \vdash (q_0, ababb, aZ_0)$

$\vdash (q_0, babb, aaZ_0)$

$\vdash (q_0, abb, aZ_0)$

$\vdash (q_0, bb, aaZ_0)$

$\vdash (q_0, b, aZ_0)$

$\vdash (q_0, \varepsilon, Z_0)$

$\vdash (q_1, Z_0)$

ACCEPT state.

Example 4.2.5 *Design a PDA for the language*

$L = \{w \mid w \in (a + b)^* \text{ and } n_a(w) > n_b(w)\}$

Solution : $n_a(w)$ means total number of a's in input string and $n_b(w)$ means total number of b's in input string. The problem states that total number of a's are more than total number of b's in input string. The logic for this PDA will be-

If we read a or b we will simply push it onto the stack. If the stack top has a symbol a and we read a, then also push it onto the stack. Same is true for b. But if we read 'a' and stack top symbol is 'b' then pop. Also if we read 'b' and stack top symbol is 'a' then pop the stack. Finally if we read ε (i.e. Complete string is read) then stack should contain a on the top. This means that total number of a's are more than total number of b's. The ID can be

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$

$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$

$$\delta(q_0, a, a) = (q_0, aa)$$

$$\delta(q_0, b, b) = (q_0, bb)$$

$$\delta(q_0, a, b) = (q_0, \varepsilon)$$

$$\delta(q_0, b, a) = (q_0, \varepsilon)$$

$$\delta(q_0, \varepsilon, a) = (q_f, a)$$

where $P = (\{q_0, q_f\}, \{a, b\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_f\})$

Simulation : We will take some string to simulate this PDA for some input string.

Consider the input aababab

$\delta(q_0, aababab, Z_0) \vdash (q_0, ababab, aZ_0)$

$\vdash (q_0, babab, aZ_0)$

$\vdash (q_0, abab, aZ_0)$

$\vdash (q_0, bab, aaZ_0)$

$\vdash (q_0, ab, aZ_0)$

$\vdash (q_0, b, aaZ_0)$

$\vdash (q_0, \varepsilon, aZ_0)$

$\vdash (q_f, a)$

Accept state.

Thus input gets accepted.

Example 4.2.6 *Design PDA for the language that accepts strings with* $n_a(w) < n_b(w)$ *where* $w \in (a + b)^*$.

Solution : This problem is similar to previous problem. The only difference is that in this problem the language requires more number of b's than total number of a's.

Hence the ID will be -

$\delta (q_0, a, Z_0) = (q_0, aZ_0)$

$\delta (q_0, b, Z_0) = (q_0, bZ_0)$

$\delta (q_0, a, a) = (q_0, aa)$

$\delta (q_0, b, b) = (q_0, bb)$

$\delta (q_0, b, a) = (q_0, \varepsilon)$

$\delta (q_0, a, b) = (q_0, \varepsilon)$

$\delta (q_0, \varepsilon, b) = (q_f, b)$ ← The b's are more in number.

Now we will **Simulate** this PDA for the input "abbab".

$\delta (q_0, abbab, Z_0) \vdash (q_0, bbab, aZ_0)$

$\vdash (q_0, bab, Z_0)$

$\vdash (q_0, ab, bZ_0)$

$\vdash (q_0, b, Z_0)$

$\vdash (q_0, \varepsilon, bZ_0)$

$\vdash (q_f, b)$

Accept state.

Example 4.2.7 *Design a PDA that accepts a string of well formed parenthesis. Consider the parenthesis is as (,), [,], { , }.*

Solution : The problem of checking well formedness of parenthesis is similar to the problem of checking equal number of a's and equal number of b's. But always (, [or { i.e. left parenthesis come first and then corresponding),] or } i.e. right parenthesis

appear. This is what is a meaning of well - formedness. The ID can be constructed as below :

Initially i.e. in state q_0 either (, { , or [will be scanned and at that time the stack will be empty. Then we will simply push it onto stack.

$\therefore \quad \delta(q_0, (, Z_0) \vdash (q_1, (Z_0)$

$\delta(q_0, [, Z_0) \vdash (q_1, [Z_0)$

$\delta(q_0, \{, Z_0) \vdash (q_1, \{Z_0)$

Further if we read more left parenthesis then we will simply push them onto the stack.

$\delta(q_1, (, () = (q_1, (()$

$\delta(q_1, [, [) = (q_1, [[)$

$\delta(q_1, \{, \{) = (q_1, \{\{)$

$\delta(q_1, (, [) = (q_1, ([)$

$\delta(q_1, (, \{) = (q_1, (\{)$

$\delta(q_1, [, () = (q_1, [()$

$\delta(q_1, \{, () = (q_1, \{()$

$\delta(q_1, \{, [) = (q_1, \{[)$

$\delta(q_1, [, \{) = (q_1, [\{)$

Thus if we read any opening parenthesis we go on pushing it onto the stack. But as soon as we read closing parenthesis or right parenthesis we start popping the stack contents.

$\therefore \quad \delta(q_1,), () = (q_1, \varepsilon)$

$\delta(q_1,], [) = (q_1, \varepsilon)$.

$\delta(q_1, \}, \{) = (q_1, \varepsilon)$

After reading the complete input string we will get ε to read but at the same time the contents of stack should be removed and it should simply contain Z_0. At this time the PDA should enter the final state and that too in state q_0, so that all steps can be repeated if needed. This transition can be as shown -

$\delta(q_1, \varepsilon, Z_0) = (q_0, Z_0)$

To summarize these steps we will write ID as -

$\delta(q_0, (, Z_0) = (q_1, (Z_0)$

$\delta(q_0, [, Z_0) = (q_1, [Z_0)$

$\delta(q_0, \{, Z_0) = (q_1, \{Z_0)$

$\delta(q_1, (, () = (q_1, (()$

$\delta(q_1, [, [) = (q_1, [[)$

$\delta(q_1, \{, \{) = (q_1, \{\{)$

$\delta(q_1, (, [) = (q_1, ([)$

$\delta(q_1, (, \{) = (q_1, (\{)$

$\delta(q_1, [, () = (q_1, [()$

$\delta(q_1, \{, () = (q_1, \{()$

$\delta(q_1, \{, [) = (q_1, \{[)$

$\delta(q_1, [, \{) = (q_1, [\{)$

$\delta(q_1,), () = (q_1, \varepsilon)$

$\delta(q_1,], [) = (q_1, \varepsilon)$

$\delta(q_1, \}, \{) = (q_1, \varepsilon)$

$\delta(q_1, \varepsilon, Z_0) = (q_0, Z_0)$

The PDA $P = (\{q_0, q_1\}, \{(, [, \{,),], \}\}, \{[, (, \{, Z_0\}, \delta, q_0, Z_0, \{q_0\})$

Let us simulate it for some input string

({ } [ ])

$\delta(q_0, (\{\}[]), Z_0) \vdash (q_1, \{\}[]), (Z_0)$

$\vdash (q_1, \}[]), \{(Z_0)$

$\vdash (q_1, []), (Z_0)$

$\vdash (q_1,]), [(Z_0)$

$\vdash (q_1,), (Z_0)$

$\vdash (q_1, \varepsilon, Z_0)$

(q_0, Z_0)

ACCEPT state.

Example 4.2.8 *Construct PDA for the language* $\{L = a^m b^m c^n \mid m, n \geq 1\}$ **AU : May-07, Marks 8**

Solution : This is the language in which all the a's are followed by equal number of b's followed by any number of c's. The simple logic that we can apply is : as we read as go on pushing each a onto the stack. As soon as we read b, pop one a from the stack. Repeat this process while reading all b's. Now when we read c, simply read c and do

nothing, because number of c's are arbitrary in given language. The Instantaneous Description (ID) for this logic can be as follows -

$$\delta(q_0, a, Z_0) = \delta(q_0, a\,Z_0)$$

$$\delta(q_0, a, a) = \delta(q_0, aa)$$

$$\delta(q_0, b, a) = \delta(q_1, \varepsilon)$$

$$\delta(q_1, b, a) = \delta(q_1, \varepsilon)$$

$$\delta(q_1, c, Z_0) = \delta(q_1, Z_0)$$

$$\delta(q_1, \varepsilon, Z_0) = \delta(q_2, Z_0)$$

Let us derive this PDA for some example.

$$\begin{aligned}\delta(q_0, aabbccc, Z_0) &\vdash \delta(q_0, abbccc, aZ_0)\\ &\vdash \delta(q_0, bbccc, aaZ_0)\\ &\vdash \delta(q_1, bccc, aZ_0)\\ &\vdash \delta(q_1, ccc, Z_0)\\ &\vdash \delta(q_1, cc, Z_0)\\ &\vdash \delta(q_1, c, Z_0)\\ &\vdash \delta(q_1, \varepsilon, Z_0)\\ &\vdash \delta(q_2, Z_0)\\ &\text{ACCEPT}\end{aligned}$$

Example 4.2.9 *Build a PDA for the language* $L=\{0^n\,1^m\,0^n \mid m,n \geq 1\}$ *by empty stack.*

Solution : In this PDA n number of 0's are followed by any number of 1's followed by n number of 0's. Hnece the logic for design of such PDA will be as follows. Push all 0's onto the stack on encountering first zeros. Then if we read 1, just do nothing. Then read 0, and on each read of 0, pop one 0 from the stack. For instance :

This scenario can be written in the ID form as

$$\delta(q_0, 0, Z_0) = \delta(q_0, 0\,Z_0)$$

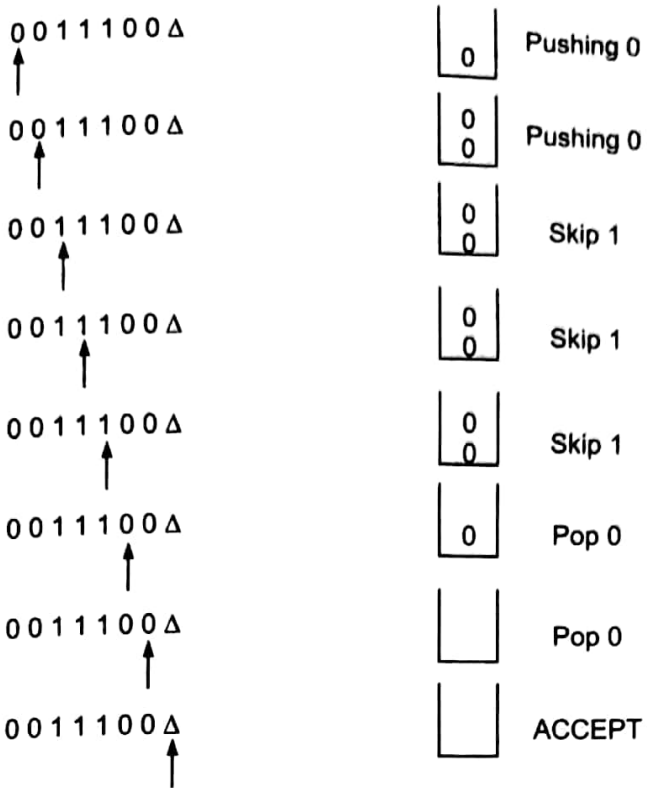
$$\delta(q_0, 0, 0) = \delta(q_0, 00)$$

$$\delta(q_0, 1, 0) = \delta(q_1, 0)$$

$$\delta(q_0, 1, 0) = \delta(q_1, 0)$$

$$\delta(q_1, 0, 0) = \delta(q_1, \varepsilon)$$

$$\delta(q_0, \varepsilon, Z_0) = \delta(q_2, Z_0) \leftarrow \text{ACCEPT state}$$



For example :

$$\delta(q_0, 0011100, Z_0) \vdash \delta(q_0, 011100, 0Z_0)$$
$$\vdash \delta(q_0, 11100, 00Z_0)$$
$$\vdash \delta(q_0, 1100, 00Z_0)$$
$$\vdash \delta(q_1, 100, 00Z_0)$$
$$\vdash \delta(q_1, 00, 00Z_0)$$
$$\vdash \delta(q_1, 0, 0Z_0)$$
$$\vdash \delta(q_1, \varepsilon, Z_0)$$
$$\vdash \delta(q_2, Z_0)$$

ACCEPT

Example 4.2.10 *Construct the PDA for* $L = \{ww^R \mid w \text{ is in } (a + b)^*\}$

AU : May-12, Marks 10; May-13, Marks 8

Solution : Let, the PDA $P = (Q, \Sigma, \Gamma, q_0, F, \delta)$

The mapping function δ is as given below -

$$\left.\begin{aligned} \delta(q_0, a, \varepsilon) &= (q_0, a) \\ \delta(q_0, b, \varepsilon) &= (q_0, b) \\ \delta(q_0, \varepsilon, \varepsilon) &= (q_f, \varepsilon) \end{aligned}\right\}$$ Pushing the elements onto stack

$\delta(q_f, a, a) = (q_f, \varepsilon)$

$\delta(q_f, b, b) = (q_f, \varepsilon)$ } Popping the elements.

$\delta(q_f, \varepsilon, \varepsilon) = (q_f, \varepsilon) \rightarrow$ ACCEPT

Simulation of abba

$\delta(q_0, \underline{a}bba, \underline{\varepsilon}) \vdash \delta(q_0, \underline{b}ba, \underline{\varepsilon}a)$

$\vdash \delta(q_0, \underline{\varepsilon}ba, \underline{\varepsilon}ba)$

$\vdash \delta(q_f, \underline{b}a, \underline{b}a)$

$\vdash \delta(q_f, \underline{a}, \underline{a})$

$\vdash \delta(q_f, \varepsilon, \varepsilon)$

ACCEPT

Example 4.2.11 *Construct a transition table for PDA which accepts the langauge* $L = \{a^{2n} b^n \mid n \geq 1\}$ *by empty stack. Trace your PDA for the input with* $n = 3$.

AU : Dec.-12, Marks 10

Solution : In this language there are twice number of a's than b's. The logic for this PDA is that - We will push all the a's onto the stack. When a single b is read, on each reading of b two a's are popped off. The ID can be constructed as follows -

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$

$\delta(q_0, a, a) = (q_0, aa)$

$\delta(q_0, b, a) = (q_1, \varepsilon)$

$\delta(q_1, \varepsilon, a) = (q_2, \varepsilon)$

$\delta(q_2, b, a) = (q_1, \varepsilon)$

$\delta(q_2, \varepsilon, Z_0) = (q_3, \varepsilon) \rightarrow$ ACCEPT

Trace for n = 3

Consider string $a^{2(3)} b^3$ = aaaaabbb

$\delta(q_0, \underline{a}aaaaabbb, \underline{Z_0}) \vdash \delta(q_0, \underline{a}aaaabbb, \underline{a}Z_0)$

$\vdash \delta(q_0, \underline{a}aaabbb, \underline{a}aZ_0)$

$\vdash \delta(q_0, \underline{a}aabbb, \underline{a}aaZ_0)$

$\vdash \delta(q_0, \underline{a}abbb, \underline{a}aaaZ_0)$

$\vdash \delta(q_0, \underline{a}bbb, \underline{a}aaaaZ_0)$

$\vdash \delta(q_0, \underline{b}bb, \underline{a}aaaaaZ_0)$

$\vdash \delta (q_1, \underline{b}bb, \underline{a}aaaaZ_0)$

$\vdash \delta (q_2, \underline{b}b, \underline{a}aaZ_0)$

$\vdash \delta (q_1, \underline{b}b, \underline{a}aaZ_0)$

$\vdash \delta (q_2, \underline{b}, \underline{a}aZ_0)$

$\vdash \delta (q_1, \underline{\varepsilon}, aZ_0)$

$\vdash \delta (q_2, \varepsilon, Z_0)$

$\vdash \delta (q_3, \varepsilon)$ ACCEPT

4.3 Languages of PDA

AU : Dec.-03,04,07,09,13,14, May-04,05,11, Marks 10

The language can be accepted by a Pushdown Automata using two approaches -

1. **Acceptance by final state :** The PDA accepts its input by consuming it and then it enters in the final state.
2. **Acceptance by empty stack :** On reading the input string from initial configuration for some PDA, the stack of PDA gets empty.

We will discuss these approaches with the help of some proofs.

4.3.1 Acceptance by Final State

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. Then the language L(P) is called the language accepted by final state.

The language L(P) can be defined as

$$L(P) = \{w \mid (q_0, w, Z_0) \vdash^* (p, \varepsilon, \alpha)\}$$

i.e. the PDA P reads the entire input w and enters in a final state p.

We will prove it with the help of some example.

Theorem

For the language $L_{wwr} = \{ww^R \mid w \text{ is in } (0 + 1)^*\}$. Consider the PDA P that accepts string x by final state if and only if x is of the form ww^R.

Proof : We will prove theorem in two parts :

if part and only if part.

Consider **if part** first. We have to prove that if $x = ww^R$ then PDA leads to accept state.

$$(q_0, ww^R, Z_0) \vdash^* (q_0, w^R, w^R Z_0) \vdash (q_1, w^R, w^R Z_0)$$
$$\vdash^* (q_1, \varepsilon, Z_0) \vdash (p, \varepsilon, Z_0)$$

That means PDA P reads w from input tape and stores it onto the stack in reverse. Then it goes to state q_1 and in q_1 state if w^R matches with w^R on stack top then finally it goes to final state p.

Now we will prove **only if** part. That is final state is achieved only if the input is of the form $x = ww^R$. Note here we are trying to check the way to enter the accept state.

When

$\delta (q_1, \varepsilon, Z_0) \vdash (p, \varepsilon, Z_0)$

i.e. to reach to state p we should have PDA to be in state q_1 with Z_0 on the top of stack with input ε. And to reach to q_1 with input ε and Z_0 at the top we should have

$$(q_1, w^R, w^R Z_0)$$

Hence we can make a more general statement as

$\delta (q_0, x, \alpha) \vdash^* (q_1, \varepsilon, \alpha)$ then x is of the form ww^R. This can be proved with the help of induction theorem.

Basis : Consider $x = \varepsilon$ then

$$x = ww^R = \varepsilon$$

then $(q_0, x, \alpha) \vdash^* (q_1, x, \beta)$ becomes

$(q_0, \varepsilon, \alpha) \vdash^* (q_1, \varepsilon, \alpha)$ is true.

Induction : Now consider $x = a_1, a_2, \ldots a_n$ for $n > 0$. There are two moves that PDA P can make.

1. For ID $(q_0, x, \alpha) \vdash (q_1, x, \alpha)$. Now being in q_1 state then PDA P can pop the stack. As PDA P reads every input symbol the stack is popped. Thus $(q_1, x, \alpha) \vdash^* (q_1, \varepsilon, \beta)$ where β will be shorter than α. It cannot be equal to α.
2. $(q_0, a_1, a_2 \ldots a_n, \alpha) \vdash (q_0, a_2 \ldots a_n, a_1 \alpha)$. We can continue this move by pushing each input symbol onto the stack. But the last move for a final state would be

$(q_1, a_n\ a_1\ \alpha) \vdash (q_1, \varepsilon, \alpha)$

In that case $a_1 = a_n$. From this we can conclude that $a_2 \ldots a_{n-1}$ is of the form yy^R for some y. We can conclude that x is of the form ww^R. This proves that PDA P accepts exactly those strings in L_{wwr}.

4.3.2 Acceptance by Empty Stack

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be the PDA then the language accepted by empty stack N(P) is defined as

$$N(P) = \{w \mid (q_0, w, Z_0) \vdash^* (p, \varepsilon, \varepsilon)\}$$

Starting from the initial configuration P reads the input w and empties its stack. The proof can be obtained with the help of some example.

Let L_{wwr} is accepted some PDA P with empty - stack. The accepting computation of P for ww^R can be

$$(q_0, ww^R, Z_0) \vdash^* (q_0, w^R, w^R Z_0) \vdash (q_1, w^R, w^R Z_0)$$
$$\vdash^* (q_1, \varepsilon, Z_0) \vdash (q_2, \varepsilon)$$

Thus the last symbol is popped off the stack as it accepts $N(P) = L_{wwr}$. Thus stack becomes empty on acceptance of complete input.

4.3.3 Equivalence of Acceptance by Final State and Empty Stack

1) If $L = N(P_1)$ for some PDA P_1 then there is a PDA P_2 such that $L = L(P_2)$. That means the language accepted by empty stack PDA will also be accepted by final state PDA.

2) If there is a language $L = L(P_1)$ for some PDA P_1 then there is a PDA P_2 such that $L = N(P_2)$. That means language accepted by final state PDA is also acceptable by empty stack PDA.

Let us have proofs for them.

1. From empty stack to final state

Theorem : If $L = N(P_1)$ for some PDA where $P_1 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ there is a PDA P_2 such that $L = L(P_2)$.

Proof : We use a new symbol X_0 which is a start symbol of PDA P_2 and marker on the bottom of the stack. When P_2 sees X_0 on top of its stack then it knows that P_1 would empty its stack on same input.

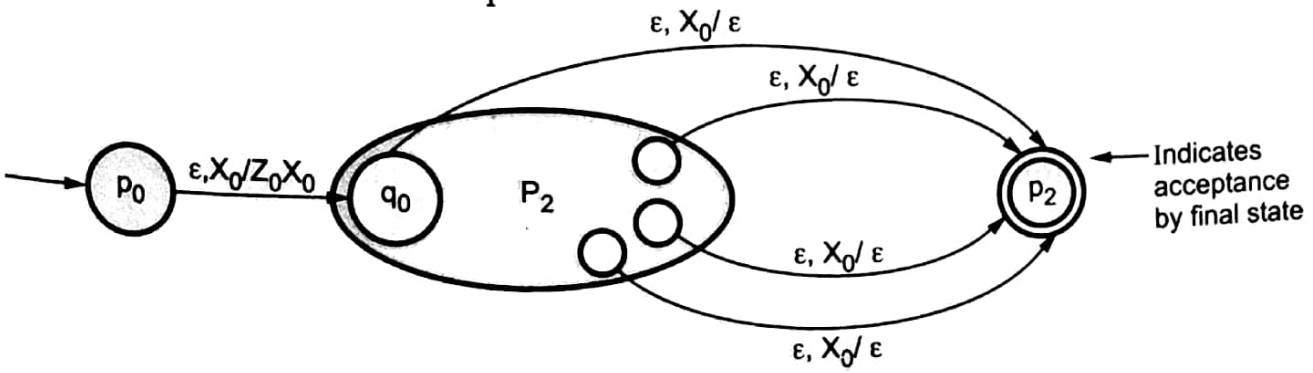


Fig. 4.3.1 Moves made by PDA P_1

There is a start state p_0 whose function is to push Z_0 onto the stack and enter state q_0 state of P_2. The P_2 simulates P_1 until the stack of P_1 is empty. It is detected by P_2 by seeing X_0 on the top of the stack. There is another new state p_2 which is accepting state of P_2. When P_1 empties its stack, P_2 moves to an accept state. The sequence of moves can be given as -

$$(p_0, w, X_0) \underset{P_2}{\vdash} (q_0, w, Z_0, X_0) \overset{*}{\underset{P_2}{\vdash}} (p, \varepsilon, X_0) \vdash (p_2, \varepsilon, \varepsilon)$$

Thus P_2 accepts w by final state.

2. From final state to empty stack

Theorem : Let $L = L(P_1)$. Construct P_2 such that $L = N(P_2)$.

Proof : P_2 uses a marker X_0 on the bottom of its stack which will act as P_2's starting stack symbol. Assume P_1 is a final state PDA and P_2 is empty stack PDA.

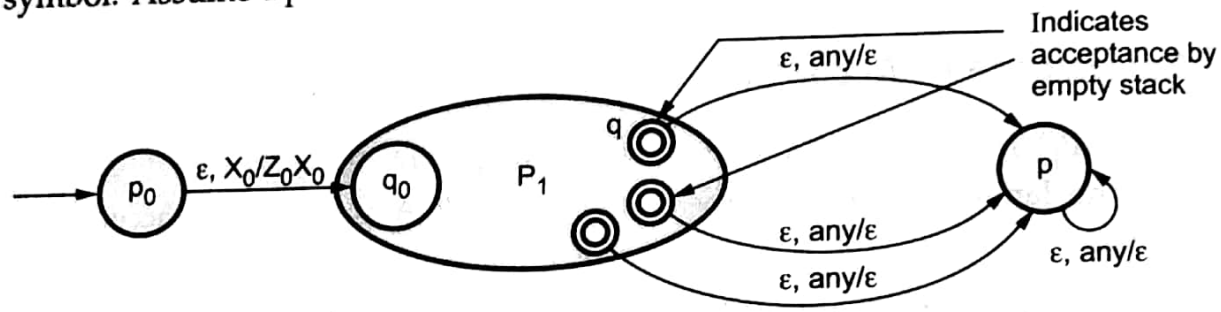


Fig. 4.3.2 Moves made by PDA P_2

p_0's sole function is to push the start symbol of PDA P_1 onto stack and move to the start state of PDA P_1. Now p is the state where the P_2's stack is emptied.

The construction of

$$P_2 = (Q \cup \{p_0, p\}, \Sigma, \Gamma, \cup \{X_0\}, \delta_N, p_0, X_0\}$$

where Q represent set of states belonging to PDA P_1.

and δ_N is defined as - [Note δ_N is a mapping function for empty stack PDA].

1. $\delta_N (p_0, \varepsilon, X_0) = \{(q_0, Z_0, X_0)\}$. Thus by pushing the start symbol of P_1 and going to start state of P_1 we will start construction of machine.
2. $\delta_N (q, a, Y)$ contains every pair that is in $\delta_F (q, a, Y)$ where q means all the states $\in Q$ and *'a'* is a input symbol either $\in \Sigma$ or ε. And Y is a stack symbol $\in \Gamma$. The δ_F is a mapping function for final state PDA.

   Thus P_2 simulates P_1.
3. When machine reaches to accepting state $q \in F$ and if we get stack symbol $Y \in \Gamma$ as X_0 then $\delta_N (q, \varepsilon, Y) = (p, \varepsilon)$.

Thus whenever P_1 accepts, P_2 can start emptying its stack without consuming any more input. Thats why in Fig. 4.3.2 transitions are shown from final states of P_1 to state p.

4. Once in state p, which is obtained only when P_1 has accepted, P_2 then pops every symbol on its stack until stack gets emptied. And thereafter no input is consumed.

Now we will prove the acceptance of string w. That w is accepted by P_2 if and only if w is in $L(P_1)$.

Key Point *In this proof we should show the design of mapping function i.e. δ_N for empty stack PDA.*

If part

Suppose, $(q_0, w, Z_0) \overset{*}{\underset{P_1}{\vdash}} (q, \varepsilon, \alpha)$ where q is some accepting state and α is a stack string.

As every transition of P_1 is a move of P_2 we will keep X_0 at the bottom of the stack for P_2. The moves of P_2 can be sequenced as follows -

$$(p_0, w, Z_0) \overset{*}{\underset{P_1}{\vdash}} (q_0, w, Z_0 X_0) \overset{*}{\underset{P_1}{\vdash}} (q, \varepsilon, \alpha X_0) \overset{*}{\underset{P_2}{\vdash}} (p, \varepsilon, \varepsilon)$$

Thus w is accepted by an empty stack PDA P_2. In other words when P_2 reaches to final state p the stack of corresponding PDA (i.e. P_2) should be empty.

Only if part

The only way P_2 can empty its stack is by entering state p. The X_0 is the bottommost symbol of P_2. Therefore X_0 is not a symbol on which P_1 has any move.

The only way by P_2 can enter state p is if the simulated P_1 enters the accepting state. The first move of P_2 will be [see Fig. 4.3.2 for better understanding].

$$\delta_N (p_0, \varepsilon, X_0) \vdash (q_0, Z_0 X_0)$$

And the accepting computation of P_2 will be -

$$(p_0, w, X_0) \underset{P_2}{\vdash} (q_0, w, Z_0 X_0) \overset{*}{\underset{P_2}{\vdash}} (q, \varepsilon, \alpha X_0) \overset{*}{\underset{P_2}{\vdash}} (p, \varepsilon, \varepsilon)$$

where q is an accept state of P_1. Similarly the computation for P_1, without X_0 on the top of stack will be -

$$(q_0, w, Z_0) \overset{*}{\underset{P_1}{\vdash}} (q, \varepsilon, \alpha)$$

That means w is accepted by final state q of P_1. Hence w is in P_1 is proved. This shows an equivalence between final state and empty stack.

Key Point *In this proof in if part we proved that PDA P_2 accept w by making moves from state p_0 to final state p and at that time stack is empty. In only if part we proved that if w is a string which is accepted by PDA P_1 then only we can construct PDA P_2 which accepts that w.*

Example 4.3.1 *Give formal pushdown automata that accepts $\{wcw^R \mid w \text{ in } (0 + 1)^*\}$ by empty stack.* **AU : Dec.-13, Marks 8**

Solution : The formal pushdown automata

$$M = (\{q_0, q_1, q_2\}, \{0, 1, c\}, \{0, 1, Z_0\}, \delta, q_0, z_0, \{q_2\}),$$ the mapping function δ is given as

$R_1 : \delta(q_0, a, Z_0) = \delta(q_0, aZ_0)$] Pushing a onto the stack

$R_2 : \delta(q_0, b, Z_0) = \delta(q_0, bZ_0)$] Pushing b onto the stack

$R_3 : \delta(q_0, a, a) = \delta(q_0, aa) \rightarrow$ Pushing a

$R_4 : \delta(q_0, b, a) = \delta(q_0, ba) \rightarrow$ Pushing b

$R_5 : \delta(q_0, a, b) = \delta(q_0, ab) \rightarrow$ Pushing a

$R_6 : \delta(q_0, b, b) = \delta(q_0, bb) \rightarrow$ Pushing b

$R_7 : \delta(q_0, c, Z_0) = \delta(q_1, Z_0) \rightarrow$ just Read c

$R_8 : \delta(q_0, c, a) = \delta(q_1, a) \rightarrow$ Pop a

$R_9 : \delta(q_0, c, b) = \delta(q_1, b) \rightarrow$ Pop b

$R_{10} : \delta(q_1, a, a) = \delta(q_1, \varepsilon) \rightarrow$ Pop a

$R_{11} : \delta(q_1, b, b) = \delta(q_1, \varepsilon) \rightarrow$ Pop b

$R_{12} : \delta(q_1, \varepsilon, Z_0) = \delta(q_2, Z_0) \rightarrow$ Empty stack i.e. **Accept state**

Review Questions

1. *If L is $N(M_1)$ (the language accepted by empty stack) for same PDA M_1 then L is $L(M_2)$ (the language accepted by final state for some PDA M_2).*
   AU : Dec.-04,14 Marks 10; May-04, Dec.-03, Marks 6; May-05, Marks 10; Dec.-07, 14, Marks 8
2. *Prove that if there exists a PDA that accepts by final state then there exists an equivalent PDA that accepts by null state.*
   AU : May-11, Marks 8

4.4 Non Deterministic Pushdown Automata

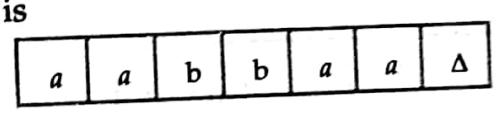
AU : May-04,05,06,09,10,11,12,13, Dec.-03,04,08,12,13, Marks 16

The non deterministic pushdown automata is very much similar to NFA. We will discuss some CFG's which accepts NPDA. The CFG which accepts deterministic PDA accepts non deterministic PDAs as well. Similarly there are some CFGs which can be accepted only by NDPA and not by DPDA. Thus NPDA is more powerful than DPDA. Let us see how to design NPDA.

Example 4.4.1 *Construct a NPDA for the language L containing all the strings which are palindrome over* $\Sigma = \{a, b\}$

Solution : Since the language consists of all the strings which are palindrome, L = {ε, *aba*, *a*, b, *aa*, bb, b*a*b, bb*a*bb, *aa*b*aa* .... }. The string can be of odd palindrome or even palindrome. The logic is very simple for constructing PDA and that is we will push a symbol onto the stack till half of the string then we will read each symbol and then perform pop operation. We will compare to see whether the symbol which is popped is similar to the symbol which is read. Whenever we reach to end of the input we expect the stack to be empty.

For example if the string is



input tape

Now the list can be divided in two halves.

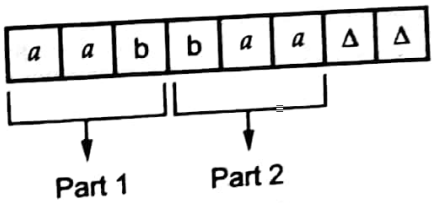


Fig. 4.4.1

We will push all the elements of part 1 onto the stack.

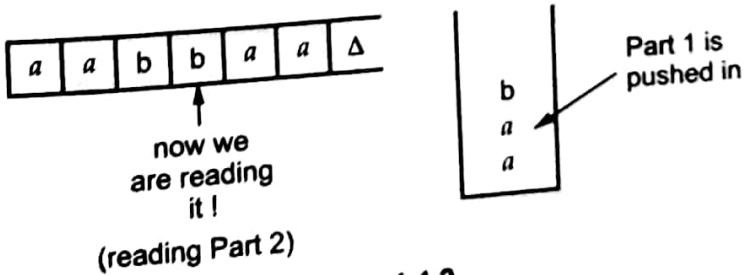


Fig. 4.4.2

Here the tape head will read the current input symbol and we will see, whether at the top of the stack same symbol is there or not.

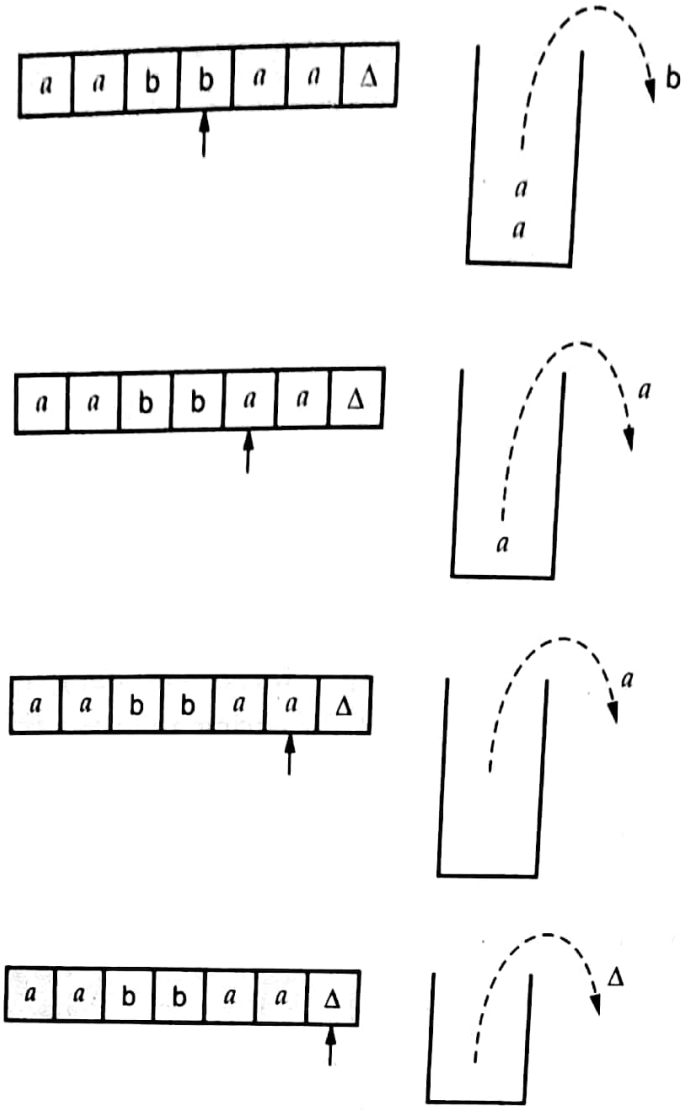


Fig. 4.4.3

The NPDA is as below -

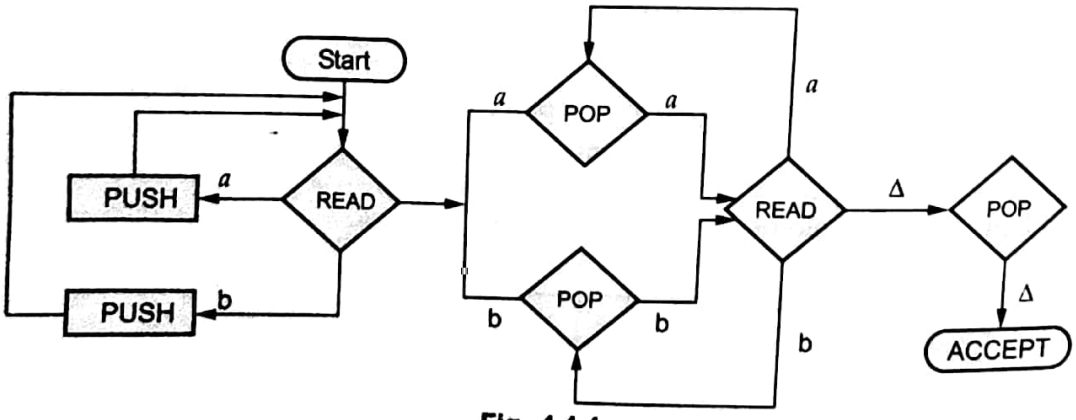


Fig. 4.4.4

The above PDA is non deterministic because for input *a* or b there are multiple paths for going to next state. We have solved this problem of palindrome for drawing a deterministic pushdown automata but there little bit manupulation is needed. We are inserting some symbol other than the input symbol exactly at a mid, so that we can get the mid of the string easily. We can obtain palindrome over $\Sigma = \{a, b\}$ by inserting X at the mid of the string. Thus let us draw this DPDA for the input set $\{a, b\}$.

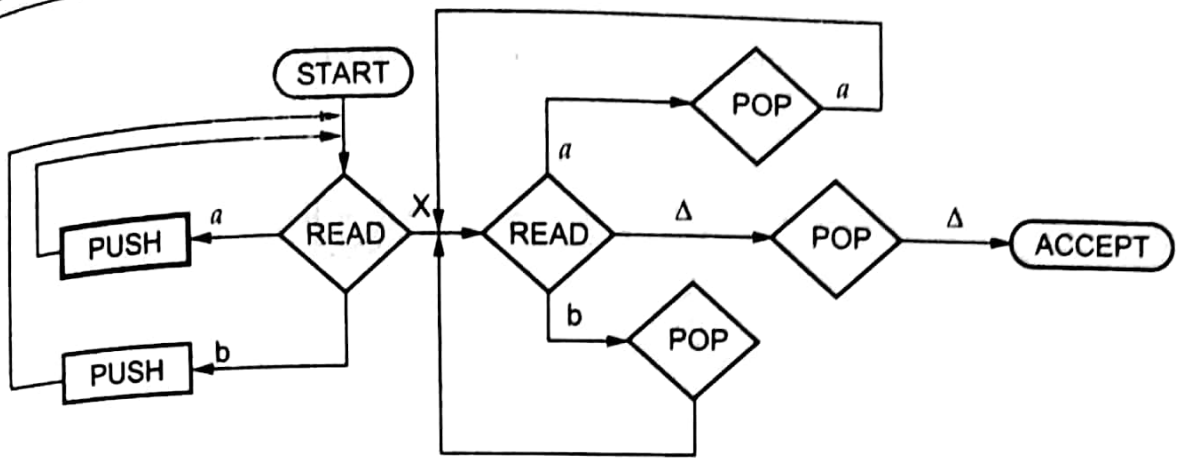


Fig. 4.4.5

Thus there exists a NPDA for a given DPDA but reverse is not always true.

4.5 Deterministic Pushdown Automata

AU : Dec.-11, Marks 10

The Deterministic Pushdown Automata (DPDA) can be defined as a collection of

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where Q is a finite set of states

Σ is a finite set of input

Γ is a finite set of stack symbols

δ is a mapping function

q_0 is initial state

Z_0 is a initial symbol in stack

F is a finite set of final states.

The machine P is deterministic if

- δ (q, a, X) has at the most **one element**.
- If δ (q, a, X) is nonempty for some a $\in \Sigma$ then δ (q, ε, X) must be empty.

In other words : There is no configuration where the machine has choice of moves. That is each transition has at most one element.

Example 4.5.1 *Consider* $L = \{w \mid w \in (a + b)^*$ *and* $n_a(w) > n_b(w)\}$. *The ID is as given below*

$\delta(q_0, a, Z_0) = (q_0, a Z_0)$

$\delta(q_0, b, Z_0) = (q_0, b Z_0)$

$\delta(q_0, a, a) = (q_0, a a)$

$\delta(q_0, b, b) = (q_0, b b)$

$\delta(q_0, a, b) = (q_0, \varepsilon)$

$\delta(q_0, b, a) = (q_0, \varepsilon)$

$\delta(q_0, \varepsilon, a) = (q_f, a)$

Is the PDA deterministic ?

Solution : We will draw the transition graph for the given PDA.

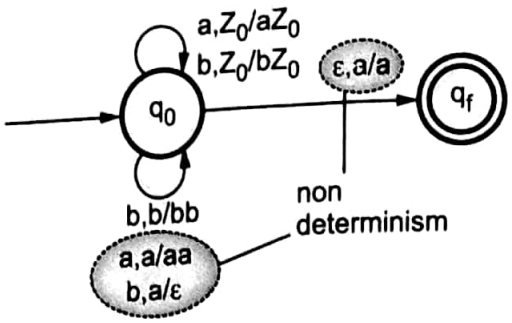


Fig. 4.5.1

We can convert it into an equivalent DPDA as -

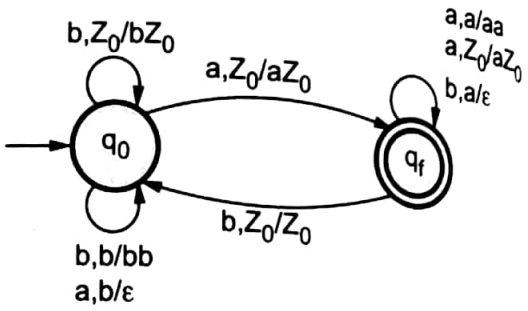


Fig. 4.5.2

i.e. $\delta(q_0, a, Z_0) = (q_f, a Z_0)$

$\delta(q_0, b, Z_0) = (q_0, b Z_0)$

$\delta(q_0, b, b) = (q_0, b b)$

$\delta(q_0, a, b) = (q_0, \varepsilon)$

$\delta(q_f, a, Z_0) = (q_f, a Z_0)$

$\delta(q_f, a, a) = (q_f, a a)$

$\delta(q_f, b, a) = (q_f, \varepsilon)$

$\delta(q_f, b, Z_0) = (q_0, Z_0)$

$\delta(q_f, \varepsilon, a) = (q_f, \varepsilon) \rightarrow$ ACCEPT

Let us see another example -

Example 4.5.2 *Design DPDA for* $L = a^n b^n$ *where* $n \geq 1$.

Solution : We will apply a very simple logic for this DPDA. When we read a we will simply push it onto the stack and as we read b simply POP corresponding a from the stack. After reading the complete input string the stack should be empty.

The ID can be -

$\delta(q_0, a, Z_0) = (q_1, a Z_0)$

$\delta(q_1, b, a) = (q_2, \varepsilon)$

$\delta(q_2, b, a) = (q_2, \varepsilon)$

$\delta(q_2, \varepsilon, Z_0) = (q_2, \varepsilon)$

The transition graph will be -

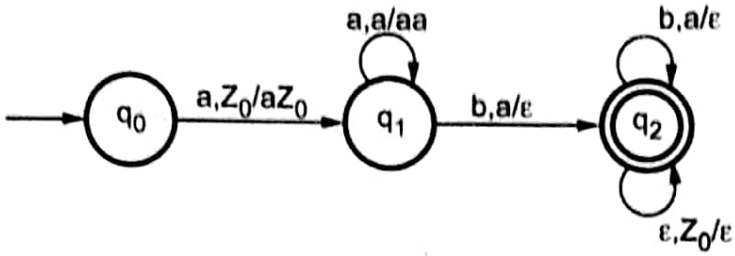


Fig. 4.5.3

Simulation

$$\delta(q_0, aabb, Z_0) \vdash (q_1, abb, aZ_0)$$
$$\vdash (q_1, bb, aaZ_0)$$
$$\vdash (q_2, b, aZ_0)$$
$$\vdash (q_2, \varepsilon, Z_0)$$
$$\vdash (q_2, \varepsilon)$$

A language L is **a Deterministic Context Free Language (DCFL)** if it is accepted by DPDA.

For example $\{L = a^n b^n \mid n \geq 1\}$ is a DCFL. Generally all regular languages can be accepted by a DPDA because every DFA is DPDA without having stack. The class of languages can be

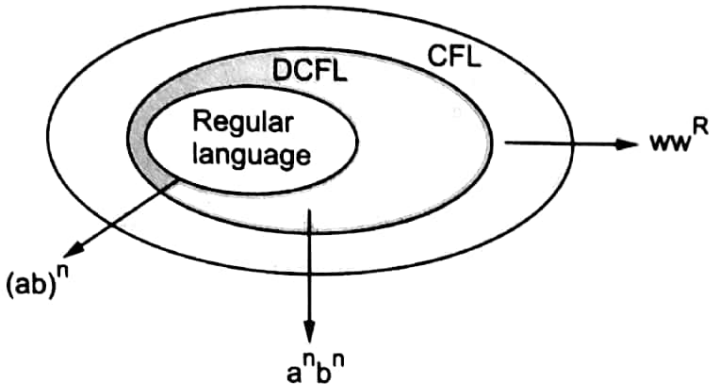


Fig. 4.5.4

- The language $ww^R \mid w \in (a, b)^*$ is not DCFL. But $wcw^R \mid w \in (a, b)^*$ is DCFL.
- All regular languages are DCFL.
- If a language is a DCFL then it has an unambiguous CFG.

Prefix Property of CFL

The language L has a prefix property if there are no two different strings x and y in L such that x is a prefix of y.

For example $\{L = wcw^r \mid w \in (0+1)^*\}$ has a prefix property. That means in L there are no such two strings with one prefix to other, unless they are same string. For instance $\underbrace{001}_{x}\ \underbrace{C100}_{y}$ and $x \neq y$.

But there are some simple languages that do not have prefix property.

For example $L = \{0\}^*$ is a language in which there are pairs which are prefix to other. Hence the languages has no prefix property.

4.6 Equivalence of Pushdown Automata and CFL

AU : Dec.-03, 04, 08, 12, 13, 14, May-04, 05, 06, 09, 10, 11, 12, 13, 14, Marks 16

4.6.1 Construction of PDA from CFG

Step 1 : Convert the given CFG to Chomsky's normal form.

Step 2 : The PDA should start by pushing start symbol onto the stack. To derive further production rules for the start symbol, we immediately perform the pop operation. It is as shown below.

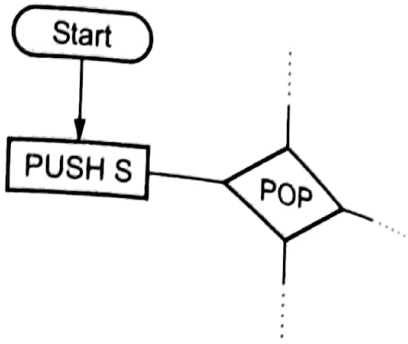


Fig. 4.6.1

Step 3 : If the production is of the form $S \rightarrow AB$, we push A and B onto the stack in reverse order. (since we get after popping the reverse order is straight ∵ reverse. reverse = straight).

So $S \rightarrow AB$

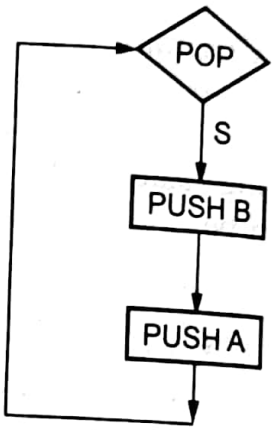


Fig. 4.6.2

Step 4 : If the production is of the form

$S \rightarrow a$ we design as

This means replacing a nonterminal S by *a*.

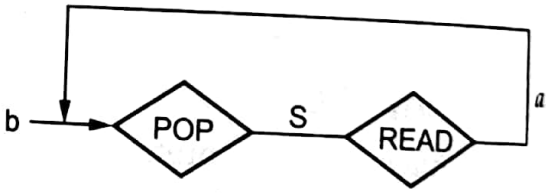


Fig. 4.6.3

Step 5 : Finally when the complete input is read from the tape, we encounter with Δ. Hence by popping Δ we get ensured with the fact that stack is also empty. This can be designed as

Fig. 4.6.4

Example 4.6.1 *Construct PDA for following grammar*

$S \rightarrow AB$

$A \rightarrow CD$

$B \rightarrow b$

$C \rightarrow a$

$D \rightarrow a$

Solution : The above grammar is already in Chomsky's normal form. So we will take the grammar as it is -

The PDA for rule 1 is

$S \rightarrow AB$

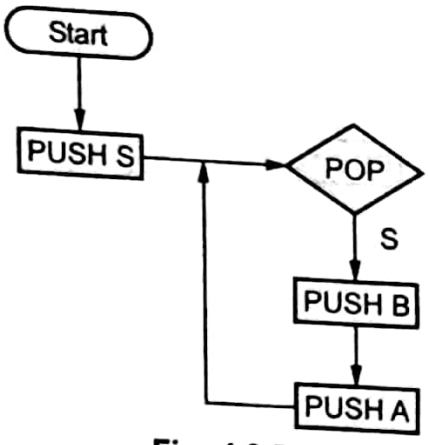


Fig. 4.6.5

$A \rightarrow CD$ can be

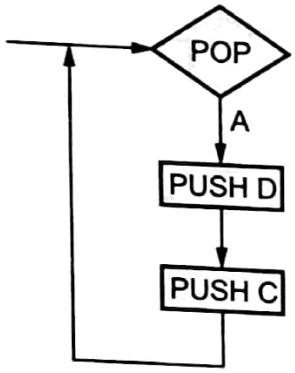


Fig. 4.6.6

$B \rightarrow b$, $C \rightarrow a$, $D \rightarrow a$ can be

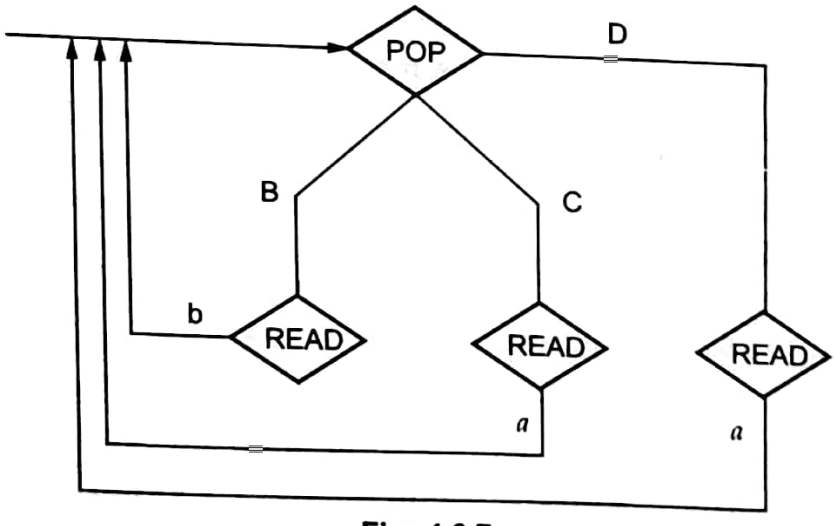


Fig. 4.6.7

Finally we could draw the complete PDA as

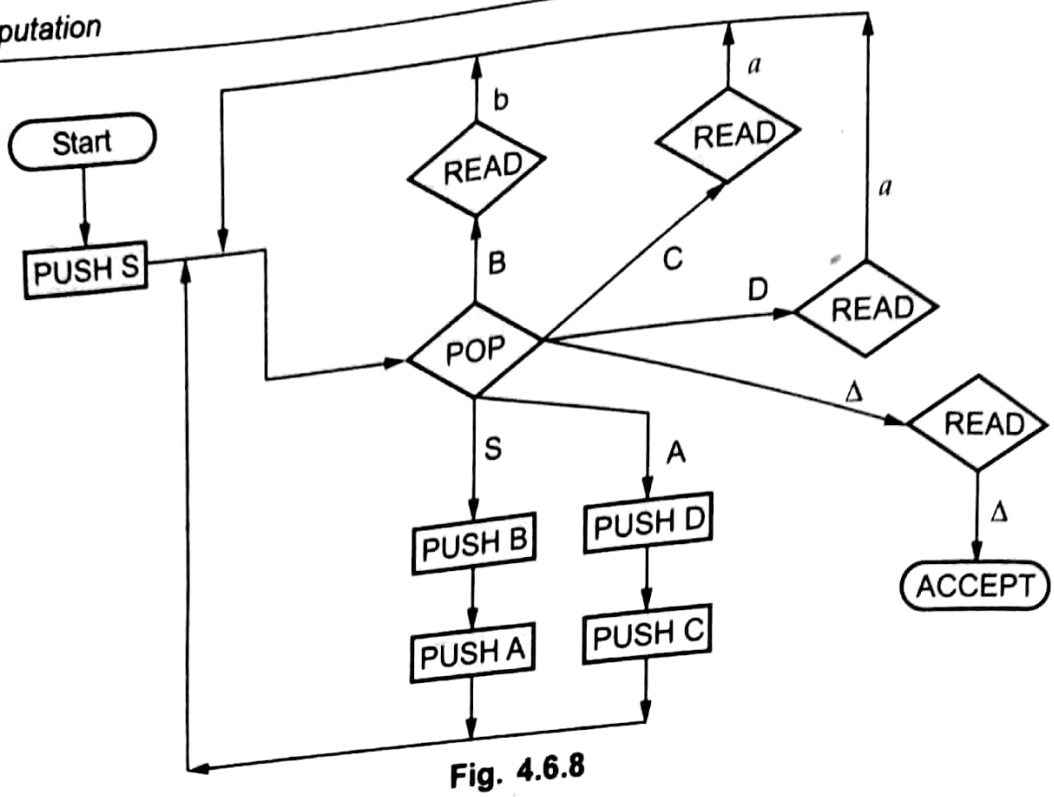


Fig. 4.6.8

Let us simulate this PDA for the string "*aa*b".

Input	Stack	Action
a a b Δ	Δ	Start
a a b Δ	S Δ	Push S
a a b Δ	Δ	POP S
a a b Δ	B Δ	Push B
a a b Δ	A B Δ	Push A
a a b Δ	B Δ	POP A
a a b Δ	D B Δ	Push D
a a b Δ	C D B Δ	Push C
a a b Δ	D B Δ	POP C
a a b Δ	D B Δ	Read *a*
a a b Δ	B Δ	POP D
a a b Δ	B Δ	Read *a*
a a b Δ	Δ	POP B
a a b Δ	Δ	Read b
a a b Δ	–	POP Δ
a a b Δ	–	Read Δ
		ACCEPT

Now to construct Instantaneous Description for this. We will first push the start symbol S onto the stack. Hence initial configuration will be -

$\delta(q, \varepsilon, Z_0) = (q, S Z_0)$

Now we will define the set of rules as

$\delta(q, w, S) = (q, AB)$

$\delta(q, w, A) = (q, CD)$ ... where w is some string

$\delta(q, b, B) = (q, \varepsilon)$

$\delta(q, a, C) = (q, \varepsilon)$

$\delta(q, a, D) = (q, \varepsilon)$

We will now simulate the string "aab" for the same

Initially push S onto the stack.

$\delta(q, \varepsilon, Z_0) = (q, S Z_0)$

Place input string w on input tape

$\delta(q, aab, S) \vdash (q, aab, AB)$

$\vdash (q, \underline{a}ab, \underline{C}AB)$

$\vdash (q, \underline{a}b, \underline{D}B)$

$\vdash (q, \underline{b}, \underline{B})$

$\vdash (q, \varepsilon)$

Accepting configuration.

Example 4.6.2 *Construct PDA for the language of any combination of 0's and 1's.*

Solution : First of all we will construct regular expression from this L.

r.e. $= (0+1)^*$

In the next step we will build the CFG for this

$S \rightarrow 0S \mid 1S \mid \varepsilon$

Now let us convert this CFG to CNF as

$S \rightarrow 0S$

$A \rightarrow 0$

Thus S becomes

$S \rightarrow AS$

$S \Rightarrow 1S$

$B \rightarrow 1$

Then, $S \rightarrow BS$

Now, $S \rightarrow \varepsilon$ is now allowed in CNF but we will take care of this rule by simply pushing and popping S. This indicates S has ε value. The rules in CNF are

$S \rightarrow AS$

$S \rightarrow BS$

$A \rightarrow 0$

$B \rightarrow 1$

As per given PDA one S after popping can be replaced by AS and another S is simply popped to have a effect of $S \rightarrow \varepsilon$. Thus you can try out the simulation of this PDA for any string of 0's and 1's.

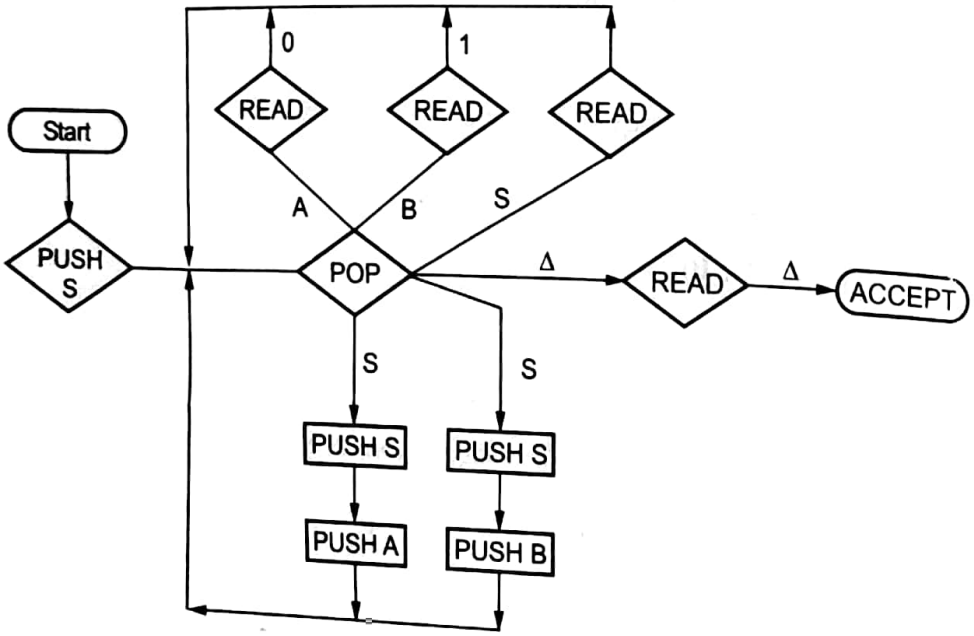


Fig. 4.6.9

Now to Construct Instantaneous Description for this we will first push the start symbol S onto the stack. Hence initial configuration will be -

$$\delta(q, \varepsilon, Z_0) = (q, SZ_0)$$

Now we will define the ID as

$$\delta(q, w, S) = (q, AS)$$

$$\delta(q, w, S) = (q, BS)$$

$$\delta(q, \varepsilon, S) = (q, \varepsilon)$$

$\delta(q,0,A) = (q,\varepsilon)$

$\delta(q,1,B) = (q,\varepsilon)$

Now we will simulate string 0101 for these rules

Initially

$\delta(q,\varepsilon,Z_0) = (q,S,Z_0)$

That is, S is pushed onto stack and string w is placed on input tape. Then,

$\delta(q,0101,S) \vdash (q,\underline{0}101,\underline{A}\,S)$

$\vdash (q,101,S)$

$\vdash (q,\underline{1}01,\underline{B}\,S)$

$\vdash (q,01,S)$

$\vdash (q,\underline{0}1,\underline{A}\,S)$

$\vdash (q,1,S)$

$\vdash (q,1,BS)$

$\vdash (q,\varepsilon,S)$

$\vdash (q,\varepsilon)$

Accepting configuration.

Example 4.6.3 *Construct PDA for the given CFG*

$S \rightarrow a\,S\,b$

$S \rightarrow a\,b$

Solution : Now we will first convert this CFG to CNF first. This is basically the language of $a^n\,b^n$ where $n \geq 1$.

$S \rightarrow a\,S\,b$ $S \rightarrow ab$	$\Rightarrow$	$S \rightarrow AR_1$ $S \rightarrow AR_2$ $R_1 \rightarrow SR_2$ $R_2 \rightarrow b$ $A \rightarrow a$

The PDA can be graphically represented as -

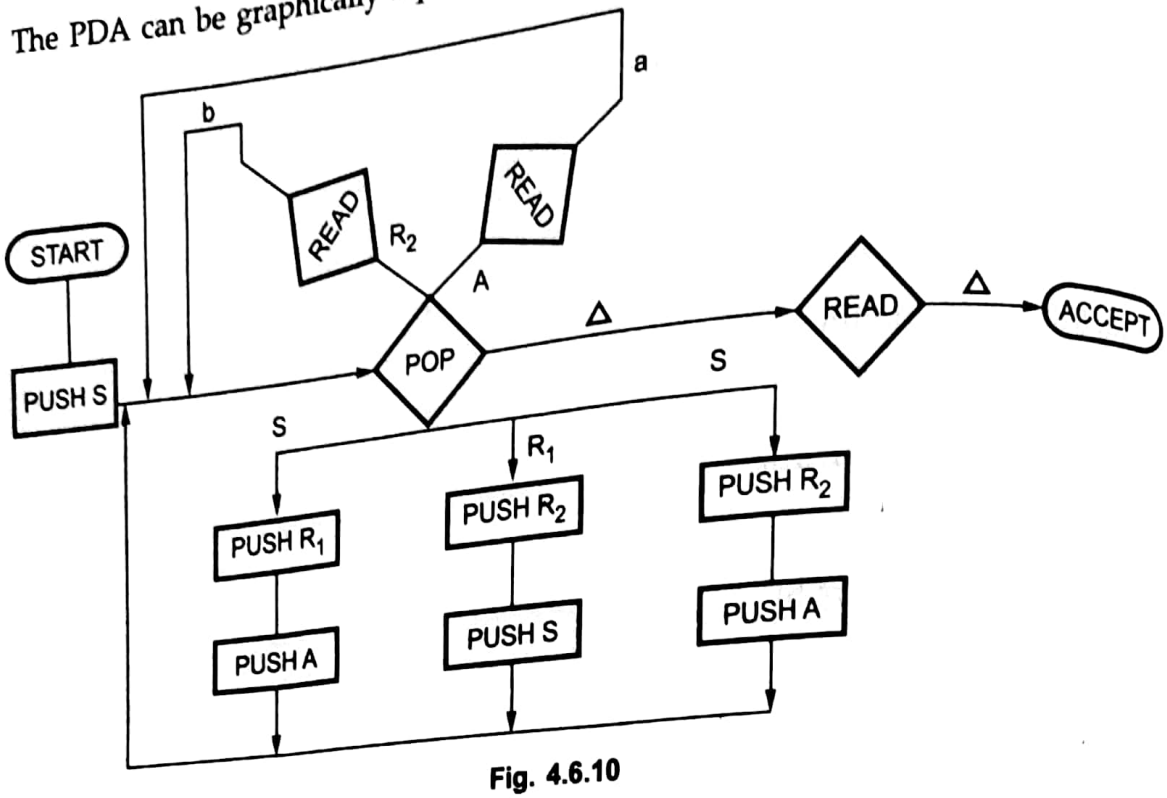


Fig. 4.6.10

The simulation for input aabb can be -

Input	Stack	Action
aabbΔ	Δ	Push S
aabbΔ	S Δ	POP S
aabbΔ	Δ	Push R_1
aabbΔ	R_1 Δ	Push A
aabbΔ	AR_1 Δ	Read a
abbΔ	R_1 Δ	POP
abbΔ	Δ	Push R_2
abbΔ	R_2 Δ	Push S
abbΔ	SR_2 Δ	POP
abbΔ	R_2 Δ	Push R_2
abbΔ	R_2R_2 Δ	Push A
abbΔ	AR_2R_2 Δ	POP
abbΔ	R_2R_2 Δ	Read a
bbΔ	R_2R_2 Δ	POP
bbΔ	R_2 Δ	Read b

bΔ	$R_2\Delta$	POP
bΔ	Δ	Read b
Δ	Δ	POP
Δ	-	Read Δ
-	-	Accept

The ID for this graphical PDA is

$$\delta(q, \varepsilon, Z_0) = (q, SZ_0)$$

Now place string w on input tape

$$\delta(q, w, S) = (q, AR_1)$$

$$\delta(q, w, S) = (q, AR_2)$$

$$\delta(q, w, R_1) = (q, SR_2)$$

$$\delta(q, b, R_2) = \delta(q, \varepsilon)$$

$$\delta(q, a, A) = \delta(q, \varepsilon)$$

Consider now input aabb for simulation -

Initially

$$(q, \varepsilon, Z_0) \vdash (q, SZ_0)$$

The start symbol is pushed onto the stack. The string aabb is placed on input tape.

$$(q, aabb, S) \vdash (q, aabb, AR_1)$$
$$\vdash (q, abb, R_1)$$
$$\vdash (q, abb, SR_2)$$
$$\vdash (q, abb, AR_2R_2)$$
$$\vdash (q, bb, R_2R_2)$$
$$\vdash (q, b, R_2)$$
$$\vdash (q, \varepsilon)$$

Accept state.

Thus input is accepted by PDA.

Alternate Method for Conversion of CFG to PDA

Step 1 : Convert the CFG to Greibach Normal Form.

Step 2 : The δ function is to be developed for the grammar of the form.

$A \rightarrow aB$ as

$\delta (q_i, a, A) \rightarrow \delta (q_i, B)$

Step 3 : Finally add the rule

$\delta (q_i, \varepsilon, Z_0) \rightarrow (q_i, \varepsilon)$

where Z_0 is the stack symbol. This is the **accept** state.

Let us solve some examples based on this method.

Example 4.6.4 *Construct PDA for the following grammar*

$S \rightarrow AB$ $A \rightarrow CD$

$B \rightarrow b$ $C \rightarrow a$

$D \rightarrow a$

Solution : The above grammar is first converted to Greibach Normal Form as follows.

$S \rightarrow CDB$ $S \rightarrow aDB$

$A \rightarrow CD \Rightarrow A \rightarrow aD$

$B \rightarrow b$ $B \rightarrow b$

$C \rightarrow a$ $C \rightarrow a$

$D \rightarrow a$ $D \rightarrow a$

Now the equivalent PDA can be -

$\delta (q_1, a, S) \rightarrow (q_1, DB)$

$\delta (q_1, a, A) \rightarrow (q_1, D)$

$\delta (q_1, b, B) \rightarrow (q_1, \varepsilon)$

$\delta (q_1, a, C) \rightarrow (q_1, \varepsilon)$

$\delta (q_1, a, D) \rightarrow (q_1, \varepsilon)$

Then add the rule

$\delta (q_1, \varepsilon, Z_0) \rightarrow (q_f, \varepsilon)$ is the accept state.

We will now simulate the string "*aa*b" for the same.

$\delta (q_1, \underline{a}ab, S) \vdash \delta (q_1, \underline{a}\ b, \underline{D}\ B)$

$\vdash \delta (q_1, \underline{b}, \underline{B})$

$\vdash \delta (q_1, \varepsilon, Z_0)$

$\vdash \delta (qf, \varepsilon)$ Accepting configuration

Example 4.6.5 *Design a PDA for recognizing the language* $\{a^m b^n c^m, n, m \geq 1\}$ *using empty stack.*

AU : Dec.-08, Marks 8

Solution : Let, L = $\{a^m b^n c^m \mid n, m \geq 1\}$ be the given language for designing this PDA, the simple logic which can be applied is - first go on pushing all a's onto the stack. When b's come, just go on reading b's simply. Then on each reading of C, just pop single a. The scenario can be.

$\delta(q_0, a, Z_0) = \delta(q_0, aZ_0)$
$\delta(q_0, a, a) = \delta(q_0, aa)$
} Push a's onto stack

$\delta(q_0, b, a) = \delta(q_1, a)$
$\delta(q_1, b, a) = \delta(q_1, a)$
} Read b's and stack will remain unchanged

$\delta(q_1, c, a) = \delta(q_1, \varepsilon)$ → Popping a on reading c.

$\delta(q_1, \varepsilon, Z_0) = \delta(q_2, Z_0)$ Acceptance by empty stack

Example 4.6.6 *Construct an unrestricted PDA equivalent ot the grammer given below :*

$S \rightarrow aAA$

$A \rightarrow aS \mid bS \mid a$

AU : Dec.-08, Marks 8

Solution : The given grammar is already in Greibach normal form. Hence the PDA can be -

$\delta(q_1, a, S) \rightarrow (q_1, AA)$
$\delta(q_1, a, A) \rightarrow (q_1, S)$
$\delta(q_1, b, A) \rightarrow (q_1, S)$
$\delta(q_1, a, A) \rightarrow (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, Z_0) \rightarrow (q_1, \varepsilon)$ ACCEPT

The simulation of abaaaa is

$\delta(q_1, \underline{a}baaaa, \underline{S}) \vdash \delta(q_1, \underline{b}aaaa, \underline{AA})$
$\vdash \delta(q_1, \underline{a}aaa, \underline{S}A)$
$\vdash \delta(q_1, \underline{a}aa, \underline{A}AA)$
$\vdash \delta(q_1, \underline{a}a, \underline{A}A)$
$\vdash \delta(q_1, \underline{a}, \underline{A})$
$\vdash \delta(q_1, \varepsilon, Z_0)$
$\vdash \delta(q_1, \varepsilon)$ ACCEPT

Example 4.6.7 *Construct an NPDA that accept the language generated by the grammar* $S \rightarrow aSbb \mid aab$.

AU : May-09, Marks 6

Solution : First we will convert the given CFG to GNF.

$S \rightarrow aSbb \Rightarrow S \rightarrow aSBB$, $B \rightarrow b$

$S \rightarrow aab \Rightarrow S \rightarrow aAB$, $A \rightarrow a$

Now consider the grammar

$S \rightarrow aSBB$

$S \rightarrow aAB$

$S \rightarrow a$

$B \rightarrow b$

The PDA can be

$\delta(q_0, a, S) = \delta(q_0, SBB)$

$\delta(q_0, a, S) = \delta(q_0, AB)$

$\delta(q_0, a, A) = \delta(q_0, \varepsilon)$

$\delta(q_0, b, B) = \delta(q_0, \varepsilon)$

$\delta(q_0, \varepsilon, z_0) = \delta(q_0, \varepsilon)$ ACCEPT.

Example 4.6.8 *Consider the GNF CNF G = ({S, T, C, D}, {a, b, c, d}, S, P) where P is :*

$S \rightarrow cCD \mid dTC \mid \varepsilon$ $\quad C \rightarrow aTD \mid c$

$T \rightarrow cDC \mid cST \mid a$ $\quad D \rightarrow dC \mid d$

Present a pushdown automaton that accepts the language generated by this grammar. Your PDA must be accepted by empty store, it must start with S on its stack and it must be based on the above grammar.

AU : May-10, Marks 16

Solution : Let PDA A = {{q}, {c, a, d}, {S, T, C, D, c, d, a}, δ, q, S, ϕ}

The production rules δ can be given as -

R1 : $\delta(q, \varepsilon, S) = \{(q, cCD), (q, dTC), (q, \varepsilon)\}$

R2 : $\delta(q, \varepsilon, C) = \{(q, aTD), (q, c)\}$

R3 : $\delta(q, \varepsilon, T) = \{(q, cDC), (q, cST), (q, a)\}$

R4 : $\delta(q, \varepsilon, D) = \{(q, dC), (q, d)\}$

R5 : $\delta(q, c, c) = \{(q, \varepsilon)\}$

R6 : $\delta(q, d, d) = \{(q, \varepsilon)\}$

R7 : $\delta(q, a, a) = \{(q, \varepsilon)\}$

} Acceptance by empty store

Simulation for string caadd

$\delta(q, \varepsilon, S) \vdash \delta(q, caadd, S)$
$\vdash \delta(q, caadd, cCD)$
$\vdash \delta(q, aadd, CD)$
$\vdash \delta(q, aadd, aTDD)$
$\vdash \delta(q, add, TDD)$
$\vdash \delta(q, add, aDD)$
$\vdash \delta(q, dd, DD)$
$\vdash \delta(q, dd, dD)$
$\vdash \delta(q, d, D)$
$\vdash \delta(q, d, d)$
$\vdash (q, \varepsilon)$ Accept

Example 4.6.9 *Design a pushdown automaton that accepts the following language.*
$L = \{ab^n cd^n | n \geq 0\}$ **AU : May-11, Marks 6**

Solution : The logic can be applied as follows -

We will simply read a, them read b and go on pushing it onto the stack. Just read c.

Then on reading each d, pop b.

The PDA will be -

$\delta(q_0, a, z_0) = \delta(q_1, z_0)$
$\delta(q_1, b, z_0) = \delta(q_1, bz_0)$
$\delta(q_1, b, b) = \delta(q_1, bb)$
$\delta(q_2, c, b) = \delta(q_3, b) \rightarrow$ just read c.
$\delta(q_3, d, b) = \delta(q_3, \varepsilon) \rightarrow$ pop b's
$\delta(q_3, \varepsilon, z_0) = \delta(q_3, z_0) \rightarrow$ Acceptance by empty stack.

Example 4.6.10 *Find the PDA equivalent to given CFG with the following productions.* **AU : Dec.-12, Marks 6**

$S \rightarrow A$, $A \rightarrow BC$, $B \rightarrow ba$ $C \rightarrow ac$.

Solution : Let the PDA A = {{q}, {a, b, c}, {S, A, B, C, a, b, c}, δ, q, S, ϕ}

The production rules δ can be given as -

$\delta(q, \varepsilon, S) = \{(q, A)\}$
$\delta(q, \varepsilon, A) = \{(q, BC)\}$
$\delta(q, \varepsilon, B) = \{(q, ba)\}$
$\delta(q, \varepsilon, C) = \{(q, ac)\}$
$\delta(q, a, a) = \{(q, \varepsilon)\}$
$\delta(q, b, b) = \{(q, \varepsilon)\}$ } ACCEPT states
$\delta(q, c, c) = \{(q, \varepsilon)\}$

Example 4.6.11 *Convert the grammar* $S \rightarrow aSb \mid A$, $A \rightarrow bSa \mid S \mid \varepsilon$ *to a PDA that accepts the same language by empty stack.*

Solution : Let, PDA $A = \{\{q\}, \{a, b\}, \{S, A, a, b\}, \delta, q, S, \phi\}$

The production rules δ can be given as -

R1 : $\delta\ (q, \varepsilon, S) = \{(q, a\ S\ b), (q, A)\}$

R2 : $\delta\ (q, \varepsilon, A) = \{(q, b\ S\ a), (q, S), (q, \varepsilon)\}$

R3 : $\delta\ (q, a, a) = \{(q, \varepsilon)\}$

R4 : $\delta\ (q, b, b) = \{(q, \varepsilon)\}$

} Acceptance by empty stack

Example 4.6.12 *Convert the grammar* $S \rightarrow 0S1 / A ; A \rightarrow 1A0 / S / \in$ *into PDA that accepts the same language by empty stack. Check whether 0101 belongs to N(M).*

AU : May-14, Marks 6

Solution : Let,

$$PDA = \{\{q\}, \{0,1\}, \{S, A, 0, 1\}, \delta, q, S, \phi\}$$

The production rules δ can be given as follows-

Rule 1 : $\delta\ (q, \varepsilon, S) = \{(q, 0S1\}, (q, A)\}$

Rule 2 : $\delta\ (q, \varepsilon, A) = \{(q, 1A0\}, (q, S)\ (q, \varepsilon)\}$

Rule 3 : $\delta\ (q, 0, 0) = \{(q, \varepsilon)\}$ Acceptance by empty stack.

Rule 4 : $\delta\ (q, 1, 1) = \{(q, \varepsilon)\}$

Simulation of 0101

$$\begin{aligned}
\delta\ (q, \varepsilon, S) &\vdash \delta (q, 0101, S) \\
&\vdash \delta (q, 101, S1) \\
&\vdash \delta\ (q, 101, 1A01) \\
&\vdash \delta\ (q, 01, A01) \\
&\vdash \delta\ (q, 01, 01) \\
&\vdash \delta\ (q, 1, 1) \\
&\vdash \delta\ (q, \varepsilon)\ \textbf{ACCEPT}
\end{aligned}$$

Proof on PDA - CFG Relationship

Theorem : If L is context free language then prove that there exists a PDA M such that L = N(M). **AU : May-05, May-04, Marks 10; May-06, Marks 12, Dec.-14, Marks 8**

Proof : Suppose w is in L(G). Then w has leftmost derivation.

$S = r_1 \underset{lm}{\Rightarrow} r_2 \underset{lm}{\Rightarrow} \ldots r_n = w$. Then we can show by method of induction that

$(q, w, S) \overset{*}{\underset{P}{-}} (q, y_i, \alpha_i)$ where y_i and α_i are representation of left-sentenial form r_i.

Basis : For $i = 1$, $r_1 = S$. Thus $x_1 = \varepsilon$, and $y_1 = w$. Since $(q, w, S) \overset{*}{-} (q, w, S)$ by 0 moves, the basis is proved.

Induction : Let

$(q, w, S) \overset{*}{-} (q, y_i, \alpha_i)$ then we have to prove $(q, w, S) \overset{*}{-} (q, y_{i+1}, \alpha_i + 1)$. The α_i is a tail that begins with A. Similarly $N_i \Rightarrow r_{i+1}$ involves replacing A by one of its production rule. Let us call this production rule as β. Then,

1. In first rule of construction of P replace A by β.
2. Then match any terminal on top of the stack with the next input symbol.

Thus we get $(q, y_{i+1}, \alpha_i + 1)$ that represents r_{i+1}.

3. Finally $\alpha_n = \varepsilon$ since the tail of $r_n = w$ is empty. Thus $(q, w, S) \overset{*}{-} (q, \varepsilon, \varepsilon)$. This proves that P accepts w by empty stack.

4.6.2 Construction of CFG from Given PDA

As per our discussion, the CFG and PDA has a strong relationship. As we have seen in the previous section, we can construct a PDA from given CFG. Similarly we can obtain CFG from given PDA.

Let, $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, q_n)$ is a PDA there exists CFG G which is accepted by PDA P. The G can be defined as

$$G = \{V, T, P, S\}$$

where S is a start symbol. T is a set of terminals and V is a set of non terminals. For getting production rules P we follow following algorithm -

Algorithm for getting production rules of CFG

1. If q_0 is start state in PDA and q_n is final state of PDA then $[q_0 Z q_n]$ becomes start state of CFG. Here Z represents the stack symbol.
2. The production rule for the ID of the form $\delta(q_i, a, Z_0) = (q_{i+1}, Z_1 Z_2)$ can be obtained as

$$\delta(q_i Z_0 q_{i+k}) \rightarrow a (q_{i+1} Z_1 q_m)(q_m Z_2 q_{i+k})$$

where q_{i+k}, q_m represents the intermediate states, Z_0, Z_1, Z_2 are stack symbols and a is input symbol.

3. The production rule for the ID of the form

$\delta(q_i, a, Z_0) = (q_{i+1}, \varepsilon)$ can be converted as $(q_i Z_0 q_{i+1}) \rightarrow a$

Let us understand this algorithm with the help of some examples.

Example for Understanding

Example 4.6.13 *Obtain CFG for the PDA given as below*

$A = (\{q_0, q_1\}, \{0,1\}, \{A, Z\}, \delta, Z, \{q_1\})$ *where* δ *is as given below*

$\delta(q_0, 0, Z) = (q_0, AZ)$

$\delta(q_0, 1, A) = (q_0, AA)$

$\delta(q_0, 0, A) = (q_1, \varepsilon)$

Solution : Now let us apply algorithm for each δ transition and obtain production rules as follows -

$\delta(q_0, 0, Z) = (q_0, AZ)$ can be converted using rule 2 of algorithm. According to rule 2 $\delta(q_i, a, Z_0) = (q_{i+1}, Z_1 Z_2)$ gives $\delta(q_i Z_0 q_{i+k}) \rightarrow a (q_{i+1} Z_1 q_m)(q_m Z_2 q_{i+k})$.

Here $q_i = q_0$, $q_{i+1} = q_0$, $a = 0$, $Z_1 = A$ and $Z_2 = Z$. So we will get -

$\delta(q_0, 0, Z) = (q_0, AZ)$ is

1. $(q_0 Z q_0) \rightarrow 0(q_0 A q_0)(q_0 Z q_0) \mid 0(q_0 A q_1)(q_1 Z q_0)$
2. $(q_0 Z q_1) \rightarrow 0(q_0 A q_0)(q_0 Z q_1) \mid 0(q_0 A q_1)(q_1 Z q_1)$

Now consider

$\delta(q_0, 1, A) = (q_0, AA)$ is

3. $(q_0 A q_0) \rightarrow 1(q_0 A q_0)(q_0 A q_0) \mid 1(q_0 A q_1)(q_1 A q_0)$
4. $(q_0 A q_1) \rightarrow 1(q_0 A q_0)(q_0 A q_1) \mid 1(q_0 A q_1)(q_1 A q_1)$

Then,

$\delta(q_0, 0, A) = (q_1, \varepsilon)$ is obtained by applying rule 3.

The rule 3 states that

if $\delta(q_i a, Z_0) = (q_{i+1}, \varepsilon)$ then

$(q_i Z_0 q_{i+1}) \rightarrow a$

Hence we get

5. $(q_0 A q_1) \rightarrow 0$

To summerize this we can write equivalent CFG as -

$(q_0 Z q_0) \rightarrow 0(q_0 A q_0)(q_0 Z q_0) \mid 0(q_0 A q_1)(q_1 Z q_0)$

$(q_0\ Z\ q_1) \rightarrow 0(q_0\ A\ q_0)(q_0\ Z\ q_1) \mid 0(q_0\ A\ q_1)(q_1\ Z\ q_1)$

$(q_0\ A\ q_0) \rightarrow 1(q_0\ A\ q_0)(q_0\ A\ q_0) \mid 1(q_0\ A\ q_1)(q_1\ A\ q_0)$

$(q_0\ A\ q_1) \rightarrow 1(q_0\ A\ q_0)(q_0\ A\ q_1) \mid 1(q_0\ A\ q_1)(q_1\ A\ q_1)$

$(q_0\ A\ q_1) \rightarrow 0$

As given in the definition of PDA A the $\{q_1\}$ is a final state. Hence according to rule 1 $[q_0\ Z\ q_n]$ is a start symbol. Hence $(q_0\ Z\ q_1)$ is a start state.

Solved Examples

Example 4.6.14 *Obtain CFG for the PDA as given below -*

$P = (\{q_0, q_1\}, \{0,1\}, \{A, Z\}, \delta, q_0, Z, \phi)$

The δ transition are -

$\delta(q_0, 0, Z) = (q_0, AZ)$

$\delta(q_0, 0, A) = (q_0, AA)$

$\delta(q_0, 1, A) = (q_1, \varepsilon)$

$\delta(q_1, 1, A) = (q_1, \varepsilon)$

$\delta(q_1, \varepsilon, A) = (q_1, \varepsilon)$

$\delta(q_1, \varepsilon, Z) = (q_1, \varepsilon)$

Solution : Consider δ transition

$\delta(q_0, 0, Z) = (q_0, A\ Z)$ ... apply rule 1

1. $(q_0\ Z\ q_0) \rightarrow 0(q_0\ A\ q_0)(q_0\ Z\ q_0) \mid 0(q_0\ A\ q_1)(q_1\ Z\ q_0)$
2. $(q_0\ Z\ q_1) \rightarrow 0(q_0\ A\ q_0)(q_0\ Z\ q_1) \mid 0(q_0\ A\ q_1)(q_1\ Z\ q_1)$

$\delta(q_0, 0, A) = (q_0, AA)$ ... apply rule 1

3. $(q_0\ A\ q_0) \rightarrow 0(q_0\ A\ q_0)(q_0\ A\ q_0) \mid 0(q_0\ A\ q_1)(q_1\ A\ q_0)$
4. $(q_0\ A\ q_1) \rightarrow 0(q_0\ A\ q_0)(q_0\ A\ q_1) \mid 0(q_0\ A\ q_1)(q_1\ A\ q_1)$

$\delta(q_0, 1, A) = (q_1, \varepsilon)$ ... apply rule 2

5. $(q_0\ A\ q_1) \rightarrow 1$

$\delta(q_1, 1, A) = (q_1, \varepsilon)$ ... apply rule 2

6. $(q_1\ A\ q_1) \rightarrow 1$

$\delta(q_1, \varepsilon, A) = (q_1, \varepsilon)$ ... apply rule 2

7. $(q_1\ A\ q_1) \rightarrow \varepsilon$

$\delta(q_1, \varepsilon, Z) = (q_1, \varepsilon)$ ... apply rule 2

8. $(q_1\ Z\ q_1) \rightarrow \varepsilon$

The rules 1 to 8 indicate the production rules CFG.

Example 4.6.15 *Construct a PDA accepting $\{a^n\ b^m\ a^n \mid m, n \geq 1\}$ by empty stack. Also construct the corresponding context free grammar accepting the same set.* **AU : Dec.-04, Marks 16**

Solution : The PDA accepting $\{a^n\ b^m\ a^n \mid m, n \geq 1\}$ is given by

$P = (\{q_0, q_1\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$

where δ is given as

1. $\delta(q_0, a, Z_0) = \{(q_0, a Z_0)\}$
2. $\delta(q_0, a, a) = \{(q_0, aa)\}$

} Push a's onto stack.

3. $\delta(q_0, b, a) = \{(q_1, a)\}$
4. $\delta(q_0, b, a) = \{(q_1, a)\}$

} Read b's and stack will remain unchanged.

5. $\delta(q_1, a, a) = \{(q_1, \varepsilon)\}$
6. $\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$

} Remaining a's are erased. Stack symbol Z_0 is erased.

Thus

$(q_0, a^n\ b^m\ a^n, Z_0) \overset{*}{\vdash} (q_1, \varepsilon, Z_0) = (q_1, \varepsilon, \varepsilon)$ is PDA.

The corresponding grammar can be constructed as follows.

Let $G = (N, T, P, S)$ where

$N = \{S, [q_0, a, q_0], [q_0, Z_0, q_0], [q_0, a, q_1], [q_0, Z_0, q_1]$
$[q_1, a, q_1], [q_1, Z_0, q_1], [q_1, a, q_0]$ and $[q_1, Z_0, q_0]\}$

$T = \{a, b\}$

The productions P are constructed as follows.

The s-productions and

1) $S \rightarrow [q_0, Z_0, q_0]$ are

2) $S \rightarrow [q_0, Z_0, q_1]$ are the two transitions from q_0.

3) For $\delta\ [q_0, a, Z_0] = [q_0, aZ_0]$ we get

$[q_0, Z_0, q_0] \rightarrow a\ [q_0, a, q_0]\ [q_0, Z_0, q_0]$

$[q_0, Z_0, q_0] \rightarrow a\ [q_0, a, q_1]\ [q_1, Z_0, q_0]$

$[q_0, Z_0, q_1] \rightarrow a\ [q_0, a, q_0]\ [q_0, Z_0, q_1]$

$[q_0, Z_0, q_1] \rightarrow a\ [q_0, a, q_0]\ [q_1, Z_0, q_0]$

4) Using rule $\delta\ (q_0, a, a) = \{(q_0, a^a)\}$ we get,

$[q_0, a, q_0] \rightarrow a\ [q_0, a, q_0]\ [q_0, a, q_0]$

$[q_0, a, q_0] \rightarrow a\ [q_0, a, q_1]\ [q_1, a, q_0]$

$[q_0, a, q_1] \rightarrow a\ [q_0, a, q_0]\ [q_0, a, q_1]$

$[q_0, a, q_1] \rightarrow a\ [q_0, a, q_1]\ [q_1, a, q_1]$

5) Using $\delta\ (q_0, b, a) = \{(q_1, a)\}$ we get,

$[q_0, a, q_0] \rightarrow b\ [q_1, a, q_0]$

$[q_0, a, q_1] \rightarrow b\ [q_1, a, q_1]$

6) $\delta\ (q_1, b, a) = \{(q_1, a)\}$ we get,

$[q_1, a, q_0] \rightarrow b\ [q_1, a, q_0]$

$[q_1, a, q_1] \rightarrow b\ [q_1, a, q_1]$

7) $\delta\ (q_1, a, a) = \{(q_1, \varepsilon)\}$ gives us

$[q_1, a, q_1] \rightarrow a$

8) $\delta\ (q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$ will give

$[q_1, Z_0, q_1] \rightarrow \varepsilon$

Thus the required CFG is constructed.

Example 4.6.16 *Let* $M = (\{q_0, q_1\}, \{0, 1\}, \{X_1, Z_0\}, \delta, q_0, Z_0, \phi)$

where δ *is given by*

$\delta\ (q_0, 0, Z_0) = \{(Q_0, XZ_0)\}$

$\delta\ (q_0, 0, X) = \{(q_0, XX)\}$

$\delta\ (q_0, 1, X) = \{(q_1, \varepsilon)\}$

$\delta\ (q_1, 1, X) = \{(q_1, \varepsilon)\}$

$\delta\ (q_1, \varepsilon, X) = \{(q_1, \varepsilon)\}$

$\delta\ (q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$

Construct CFG $G = (V, T, P, S)$ *generating* $N(M)$.

AU : Dec.-03, Marks 10; May-05,14, Marks 10

Solution : We have to construct CFG G = (V, T, P, S) for the given PDA. The V represents set of non-terminals such that

$$V = \{S, [q_0, X, q_0], [q_0, X, q_1], [q_1, X, q_0], [q_1, X, q_1]\ [q_0, Z_0, q_0],$$
$$[q_0, Z_0, q_1], [q_1, Z_0, q_0], [q_1, Z_0, q_1]\}$$

The T represents set of terminals. Hence,

$$T = \{0, 1\}$$

The production rules can be constructed from the δ function. For the start state production S we can write,

$$S \rightarrow [q_0, Z_0, q_0]$$

$$S \rightarrow [q_0, Z_0, q_1]$$

Now we can write the productions for $[q_0, Z_0, q_0]$ and $[q_0, Z_0, q_1]$.

1) $[q_0, Z_0, q_0] \rightarrow 0\ [q_0, X, q_0]\ [q_0, Z_0, q_1]$ and
$[q_0, Z_0, q_0] \rightarrow 0\ [q_0, X, q_1]\ [q_1, Z_0, q_1]$

2) $[q_0, Z_0, q_1] \rightarrow 0\ [q_0, X, q_0]\ [q_0, Z_0, q_1]$
$[q_0, Z_0, q_1] \rightarrow 0\ [q_0, X, q_1]\ [q_1, Z_0, q_1]$

Similarly

3) $[q_0, X, q_0] \rightarrow 0\ [q_0, X, q_0]\ [q_0, X, q_0]$
$[q_0, X, q_0] \rightarrow 0\ [q_0, X, q_1]\ [q_1, X, q_0]$
$[q_0, X, q_1] \rightarrow 0\ [q_0, X, q_0]\ [q_0, X, q_1]$
$[q_0, X, q_1] \rightarrow 0\ [q_0, X, q_1]\ [q_1, X, q_1]$

For the rule $\delta\ (q_0, 0, X) = \{(q_0, XX)\}$

4) For the rule $\delta\ (q_0, 1, X) = \{(q_1, \varepsilon)\}$ we get,
$[q_0, X, q_1] \rightarrow 1$

5) For the rule $\delta\ (q_1, 1, X) = \{(q_1, \varepsilon)\}$
$[q_1, X, q_1] \rightarrow 1$

6) For the rule $\delta\ (q_1, \varepsilon, X) = \{(q_1, \varepsilon)\}$
$[q_1, X, q_1] \rightarrow \varepsilon$

7) For the rule $\delta\ (q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$ we get,
$[q_1, Z_0, q_1] \rightarrow \varepsilon$

There is no production for $[q_1, 1, q_0]$ and $[q_1, Z_0, q_0]$.

Review Questions

1. *If L is N(M) for some PDA M, then prove that L is context free language.*

AU : May-06, Marks 16

2. *Discuss the equivalence between PDA and CFG.*

AU : May-12, Marks 6; May-13, Marks 16; Dec.-13, Marks 10

4.7 Pumping Lemma for CFL

AU : May-07, 08, 10, 13, 14, Dec.-07, 09, 10, 11, 12, 13, 14, Marks 16

The pumping lemma for regular sets states that every sufficiently long string in a regular set contains a short substring that can be pumped. In other words, if a long string is given and if we push or pump any number of substrings in any number of times then we always get a regular set. According to pumping lemma for CFL's there are always two short substrings which are close to each other and these both the substrings can be repeated as many times as required.

Lemma : Let L be any context free language, then there is a constant n, which depends only upon L, such that there exist a string $w \in L$ and $|w| \geq n$ where $w = pqrst$ such that

1. $|qs| \geq 1$
2. $|prs| \leq n$ and
3. For all $i \geq 0$ $p q^i r s^i t$ is in L.

Proof : This pumping lemma states that if there is a language L which is without unit productions and without a null production and there exist w where $w \in L$. The string w can be derived by a context free grammar G. The G be a grammar which is in Chomsky's Normal Form. The grammar G generates language L. For the string w, we can obtain a parse tree which derives the string w. Then if the length of the path to w is less than equal to i then the length of the word w is less than or equal to 2^{i-1}. We can prove this by induction step.

Fig. 4.7.1

Basis : If $i = 1$

Let G contains the rule $S \rightarrow a$ where length of the derived string is 1 i.e. $i = 1$. Now according to the rule the word length should be $\leq 2^{i-1}$ i.e. $2^0 = 1$. Observe that we have a word 'a' which is of length 1. Also observe that the grammar G is in Chomsky's normal form. This language is regular since $|w| = |pqrst| = 1$.

Induction step : Let w be a string which is derived by grammar G. Let k be a variable such that $n = 2^k$, $|w| \geq n$ then $|w| > 2^{k-1}$ while deriving w string we may get some non-terminals of CFG - G can be repeated for any number of times and will give the string w. If we pump the substrings to w such that the path length of this newly formed string w′ (w + Pumped string = w′) is i and the word length of w′ is $\leq 2^{i-1}$ then the grammar G deriving w′ is called a regular grammar. The necessary condition is that grammar G is in Chomsky's Normal Form.

Let us consider a grammar

$G = (\{A, B, C\}, \{a\}, \{A \rightarrow BC$

$B \rightarrow BA$

$C \rightarrow BA$

$A \rightarrow a$

$B \rightarrow b\}, A)$

Thus

$$A \overset{*}{\Rightarrow} bba = w$$

i.e. Path length i = 3

$|w| \leq 2^{i-1}$ i.e. $3 \leq 2^2$

If we pump a substring into w which satisfies the condition as $i \leq |w| \leq 2^{i-1} \leq n$ the grammar producing string w is a regular grammar.

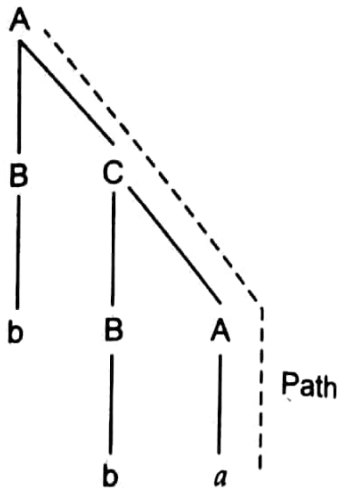


Fig. 4.7.2

Example 4.7.1 *Use pumping lemma to prove that the following is not CFL.*

$\{a^n b^m a^n b^{n+m} \mid m, n \geq 0\}$

Solution : Consider $L = \{a^n b^m a^n b^{n+m}\}$ can be written as

$$L = \{a^n b^m a^n b^n b^m\}$$

Assume that given language L is CFL. Now consider following cases.

Case 1 : Let $z = a^n b^m a^n b^n b^m \in L$

According to pumping lemma if we pump some strings to z then also $z \in L$. S language is called CFL.

We will map z = pqrst to given z

$$\therefore \quad z = \overbrace{a \ldots a}^{n} \overbrace{b \ldots b}^{m} \overbrace{a \ldots a}^{n} \overbrace{b \ldots b}^{n} \overbrace{b \ldots b}^{m}$$

p qrs t

By pumping lemma, $z = pq^i rs^i t \in L$

Let i = 2 then z becomes

$$z = pqqrsst$$

$$\therefore \quad z = a^n b^{m+t} a^{n+u} b^{n+v} b^m \notin a^n b^m a^n b^n b^m$$

Hence $z \notin L$.

Case 2 : Strings of a's only

Consider,

$$z = \overbrace{a\ldots a}^{n}\overbrace{b\ldots b}^{m}\overbrace{a\ldots a}^{n}\overbrace{b\ldots b}^{n}\overbrace{b\ldots b}^{m}$$

pqrs t

By pumping lemma, $z = pq^irs^it$. If i = 2 then $z = pq^2rs^2t$. Then z becomes

$$z = a^{n+t+u}b^m a^n b^n b^m \notin a^n b^m a^n b^n b^m$$

Case 3 : Strings of b's only.

Consider,

$$z = \overbrace{a\ldots a}^{n}\overbrace{b\ldots b}^{m}\overbrace{a\ldots a}^{n}\overbrace{b\ldots b}^{n}\overbrace{b\ldots b}^{m}$$

p qrst

By pumping lemma, $z = pq^irs^it$. If i = 2 then $z = pq^2rs^2t$. Then z becomes

$$z = a^n b^m a^n b^n b^{m+t+u} \notin a^n b^m a^n b^n b^m$$

From all these cases, our assumption of L being CFL is wrong. Hence it is proved that the given language $L = \{a^n b^m a^n b^{n+m}\}$ is not CFL.

Example 4.7.2 *Show that the language* $L = \{a^i b^j c^k \mid i<j \text{ and } i<k\}$ *is a non context free language.*

Solution : Let us prove this statement by method of contradiction. Assume L is a context free language. And w be any string such that $w \in L$.

Let, w be $w = p\,q\,r\,s\,t$

Let, $|qs| \geq 1$ and $|p\,r\,s| \leq n$ and the prs can contain at the most strings of *a*'s and c's. The string $p\,q^i\,r\,s^i\,t$ contains additional occurrences of q's and s's.

As per the condition mentioned in a lemma, $|qs| \geq 1$ this means that either q or s contains at least one a then qs can not contain any c's and as per the language L the number of c's has to be more than number of *a*'s. Therefore the $w \notin L$. Similarly if $w = p\,q^0\,r\,s^0\,t$ then also there are fewer occurrences of c's than *a*'s which cannot be our L. Hence the assumption which we had made that L is a context free language is wrong.

Example 4.7.3 *Show that* $L = \{a^n b^n c^n \mid n \geq 0\}$ *is not a context free language.*

PU : May-07,14, Dec.-10,14

Solution : Let us assume L is a context free language and $w \in L$. The w can be written as -

$w = a^n b^n c^n$ which can also be written as -

$$w = \overbrace{a \ldots a}^{n} \overbrace{b \ldots b}^{n} \overbrace{c \ldots c}^{n}$$

Consider following cases -

Case 1 : Let,

$$w = \underbrace{\overbrace{a \ldots a}^{n}}_{p} \underbrace{\overbrace{b \ldots b}^{n}}_{qrs} \underbrace{\overbrace{c \ldots c}^{n}}_{t}$$

By pumping lemma, $w = pq^i rs^i t \in L$ when $i>=0$. If we assume $i = 0$ then

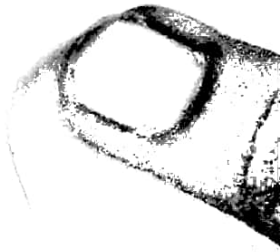
$$w = \underbrace{\overbrace{a \ldots a}^{n}}_{p} \underbrace{\overbrace{b \ldots b}^{n}}_{q^0 rs^0} \underbrace{\overbrace{c \ldots c}^{n}}_{t}$$

$$\therefore \quad w = a^n b^{n-m-k} c^n \notin a^n b^n c^n$$

Hence $w \notin L$

Case 2 : Let,

$$w = \underbrace{\overbrace{a \ldots a}^{n}}_{pqrs} \underbrace{\overbrace{b \ldots b}^{n} \overbrace{c \ldots c}^{n}}_{t}$$



By pumping lemma $w = pq^i rs^i t \in L$. If $i = 2$ then

$$w = \underbrace{\overbrace{a \ldots a}^{n+m+k}}_{pq^2 rs^2} \underbrace{\overbrace{b \ldots b}^{n} \overbrace{c \ldots c}^{n}}_{t}$$

Thus $w = a^{n+m+k} b^n c^n \notin L$.

From above cases we conclude that our assumption of L being a CFL is wrong. Hence it is proved that $L = \{a^n b^n c^n\}$ is not a context free language.

Example 4.7.4 *Determine whether the language given by* $L = \left\{a^{n^2} \mid n \geq 1\right\}$ *is context free or not.*

AU : May-08, Marks 16

Solution : We assume that given language is CFL.

Let Z be some string ∈ L. Then map

$$Z = pqrst$$

$$= a^{n^2}$$

$$= \overbrace{a.....a}^{n \times n}$$

By pumping lemma, $Z = pq^i rs^i t \in L$

Case 1 : Let,

$$Z = pq^2rs^2t$$

$$Z = \underbrace{\overbrace{a.....a}^{n^2-m}}_{pq^irs^i} \underbrace{\overbrace{a.....a}^{m}}_{t}$$

If $i = 2$ then,

$$Z = \overbrace{a.....a}^{n^2-m+t} \overbrace{a.....a}^{m}$$

i.e. $|Z| = |n^2 - m + t + m|$

But $n^2 - m + t + m \notin n^2$

Hence our assumption of L being CFG is wrong.

Case 2 :

Let $Z = \underbrace{\overbrace{a.....a}^{n^2-m}}_{pq^irs^i} \underbrace{\overbrace{a.....a}^{m}}_{t}$

If $i = 0$ then $Z = pr$ then

$$Z = \overbrace{a.....a}^{n^2-m-t} \overbrace{a.....a}^{m}$$

i.e. $|Z| = |n^2 - t|$

But $n^2 - t \notin n^2$

Hence our assumption of L being CFG is wrong. This proves that given language is not a CFG.

Example 4.7.5 *Define pumping Lemma for context free languages. Show that* $L = \{a^i b^j c^k : i < j < k\}$ *is not context free language.* **AU : May-10, Marks 10**

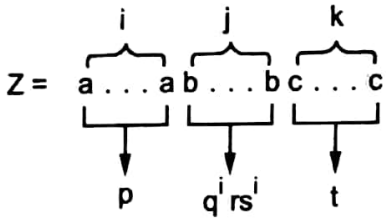
Solution : Pumping Lemma - Refer section 4.7.

The languages $\{ L = a^i b^j c^k \mid i < j < k \}$ is given.

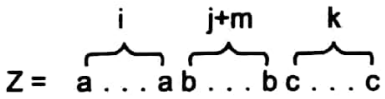
If we consider some string $Z \in L$, such that

$$Z = a^i b^j c^k$$

According to pumping lemma, if we pump some strings in between then also the $Z \in L$.

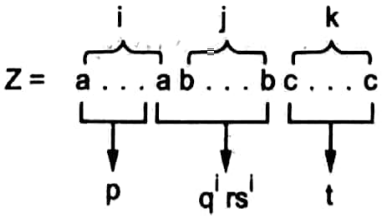


We consider following cases assuming L as CFL.



Case 1 : Strings of b's only

Let,



If $i = 2$, then $Z = pqqrsst$ then

But then $j+m > k$ can be possible. Hence Z will not belong to L.

Case 2 : Strings of a's and b's

If $i = 2$ then $Z = pqqrsst$. then

$$Z = a^i (ab)^m b^{j-m} c^k \notin a^i b^j c^k \notin L$$

Thus our assumption of L being CFL is wrong. Hence given L is not CFL.

Example 4.7.6 *Prove that* $L = \{a^m b^{m+1} c^{m+2} \mid m > 0\}$ *is not a CFL.*

Solution : We assume that given language is CFL. Let Z be some string and $Z \in L$. We can write Z as

$$Z = pqrst$$
$$= a^m b^{m+1} c^{m+2}$$

By pumping Lemma, $Z = pq^i rs^i t$

Case 1 : $i = 2$

$$Z = pq^2 rs^2 t$$
$$= \underbrace{\overbrace{a...ab...b}^{m+m+1+n}}_{pq^2rs^2} \underbrace{\overbrace{c...c}^{m+2}}_{t}$$

$$Z = a^{m+n} b^{m+1} c^{m+2} \notin L$$

Case 2 : $i = 0$

$$Z = pq^0 rs^0 t$$
$$= prt$$
$$Z = \underbrace{\overbrace{a...a}^{m-n}}_{pq} \underbrace{\overbrace{b...b}^{m+1}}_{r} \underbrace{\overbrace{c...c}^{m+2}}_{t} \notin L$$

From above two cases, our assumption of L being CFL is wrong. Hence given L is not a CFL is proved.

Review Questions

1. *State the pumping lemma for CFLs. What is its main application ? Give two examples.* **AU : Dec.-11, Marks 8**
2. *State and prove the pumping lemma for CFL.* **AU : Dec.-12, Marks 10**
3. *Explain in detail about pumping lemma for CFL.* **AU : May-13, Marks 8**

4.8 Properties of Context Free Languages

AU : Dec.-07, 09, 10, 13, May-12, 13, Marks 6

We will discuss closure properties of context free languages. In the sense, we will, check which are those properties for them CFLs are closed under. The context free languages are closed under some operation means after performing that particular

operation on those CFLs the resultant language is context free language. These properties are as below

1. The context free languages are closed under union.
2. The context free languages are closed under concatenation.
3. The context free languages are closed under kleen closure.
4. The context free languages are not closed under intersection.
5. The context free languages are not closed under complement.

We will discuss the above mentioned closure properties of CFL with the help of proofs and examples.

Theorem 1 : If L_1 and L_2 are context free languages then $L = L_1 \cup L_2$ is also context free. That is, the CFLs are closed under union.

Proof : We will consider two languages L_1 and L_2 which are context free languages. We can give these languages using context free grammars G_1 and G_2 such that $G_1 \in L_1$ and $G_2 \in L_2$. The G_1 can be given as $G_1 = \{V_1, \Sigma, P_1, S_1\}$ where P_1 can be given as

$$P_1 = \{$$
$$S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S B_1 \mid \varepsilon$$
$$A_1 \rightarrow a$$
$$B_1 \rightarrow b$$
$$\}$$

Here $V_1 = \{S_1, A_1, B_1\}$ and S_1 is a start symbol.

Similarly, we can write $G_2 = \{V_2, \Sigma, P_2, S_2\}$

$N_2 = \{S_2, A_2, B_2\}$ and S_2 is a start symbol.

P_2 can be given as :

$$P_2 = \{$$
$$S_2 \rightarrow a A_2 A_2 \mid b B_2 B_2$$
$$A_2 \rightarrow b$$
$$B_2 \rightarrow a$$
$$\}$$

Now $L = L_1 \cup L_2$ gives $G \in L$. This G can be written as

$G = \{V, \Sigma, P, S\}$

$V = \{S_1, A_1, B_1, S_2, A_2, B_2\}$

$P = \{P_1 \cup P_2\}$

S is a start symbol

$P = \{ \quad S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S B_1 \mid \varepsilon$

$A_1 \rightarrow a$

$B_1 \rightarrow b$

$S_2 \rightarrow aA_2A_2 \mid bB_2B_2$

$A_2 \rightarrow b$

$B_2 \rightarrow a$

$\}$

Thus grammar G is a context free grammar which produces languages L which is context free language.

Theorem 2 : If L_1 and L_2 are two context free languages then L_1L_2 is CFG. That means context free languages are closed under concatenation.

Proof : Let L_1 is a context free language which can be represented by a context free grammar G_1, such that $G_1 \in L_1$ and

$G_1 = \{V_1, \Sigma, P_1, S_1\}$

$V_1 = \{S_1, A_1, B_1\}$

$\Sigma = \{a, b\}$

S_1 is a start symbol and P_1 is a set of production rules,

$P_1 = \{S_1 \rightarrow A_1 S_1 A_1 \mid B_1 S_1 B_1 \mid \varepsilon$

$A_1 \rightarrow a$

$B_1 \rightarrow b$

$\}$

Similarly, L_2 is a context free language which can be represented by a context free grammar G_2, such that $G_2 \in L_2$ and

$G_2 = \{V_2, \Sigma, P_2, S_2\}$

$V_2 = \{S_2, A_2, B_2\}$

$\Sigma = \{a, b\}$

S_2 is a start symbol and P_2 is a set of production rules,

$P_2 = \{S_2 \rightarrow aA_2A_2 \mid bB_2B_2$

$A_2 \rightarrow b$

$B_2 \rightarrow a$

$\}$

Now $L = L_1 L_2$ can be obtained by G such that $G = G_1 \cdot G_2$. Therefore

$G \leftarrow \{V, \Sigma, P, S\}$

$V = \{S, S_1, A_1, B_1, S_2, A_2, B_2\}$

where S is a start symbol. The production rules, P can be given as

$P = \{ \; S \to S_1 \mid S_2$

$S_1 \to A_1 S_1 A_1 \mid B_1 S_1 B_1 \mid \varepsilon$

$A_1 \to a$

$B_1 \to b$

$S_2 \to aA_2A_2 \mid bB_2B_2$

$A_2 \to b$

$B_2 \to a$

$\}$

As grammar G is context free grammar the language L produced by G is also context free language. Hence context free languages are closed under concatenation.

Theorem 3 : If L_1 is context free language then L_1^* is also context free. That means CFL is closed under kleen closure.

Proof : Let, L_1 be a context free language represented by G_1 such that $G_1 \to \varepsilon\ L_1$.

The CFG G_1 can be given as

$G_1 = \{V_1, \Sigma, P_1, S_1\}$ where S_1 is a start symbol

$P_1 = \{S_1 \to A_1S_1A_1 \mid B_1S_1B_1 \mid \varepsilon$

$A_1 \to a$

$B_1 \to b$

$\}$

Now $L = L_1^*$ can be represented by a grammar G such that

$G = \{ (V, \Sigma, P, S)$

$V = \{S, S_1, A_1\ B_1\}$

and $P = S \to S_1\ S \mid \varepsilon$

$S_1 \to A_1\ S_1\ A_1 \mid B_1\ S_1\ B_1$

$A_1 \to a$

$B_1 \to b$

$\}$

Thus grammar G is a context free grammar and language L produced by G is also context free language. Hence context free language are closed under kleen closure.

Theorem 4 : If L_1 and L_2 are two CFLs then $L = L_1 \cap L_2$ may be CFL or may not be CFL. That means L is not closed under intersection.

Proof : Let, $L_1 = \{0^n 1^n 2^i \mid n \geq 1 , i \geq 1\}$

$L_2 = \{0^n 1^n 2^n \mid n \geq , i \geq 1\}$

The grammar for L_1 is

$S \rightarrow AB$

$A \rightarrow 0A_1 \mid 01$

$B \rightarrow 2B \mid 2$

Similarly L_2 can be represented by grammar.

$S \rightarrow AB$

$A \rightarrow 0A \mid 0$

$B \rightarrow 1B2 \mid 12$

Now if we try to obtain

$L = L_1 \cap L_2$ then we get sometimes context free languages and sometimes non context free languages. Thus we can say that CFLs are not closed under intersection.

Theorem 5 : If L_1 is a CFL then L_1' may or may not be CFL. That means CFL is not closed under complement.

Proof : Let L_1 and L_2 are two CFLs. We will assume that complement of a context free language is a CFL itself. Hence L_1' and L_2' both are CFLs. We can also state that $(L_1' \cup L_2')$ is context free (since CFLs are closed under union). But $(L_1' \cup L_2') = L_1 \cap L_2$ i.e. $L = L_1 \cap L_2$ may or may not be CFL. The L_1 and L_2 are arbitrary CFLs, there may exist L_1' and L_2' which are not CFL. Hence complement of certain language may be context free or may not be. Therefore we can say that CFL is not closed under complement operation.

Example 4.8.1 *Show that context free languages are closed under union operation but not under intersection.* **AU : Dec.-07, Marks 4**

Solution : We can show this in two steps.

Step 1 : Context free languages are closed under union.

Proof : Let L_1 and L_2 be two context free languages and G_1 and G_2 be context free grammars such that $G_1 \in L_1$ and $G_2 \in L_2$ G_1 can be $= \{V_1, \Sigma, P_1, S_1\}$ where P_1 can be

$P_1 = \{$

$$S_1 \rightarrow A_1 S_1 A_1 | B_1 S B_1 | \varepsilon$$
$$A_1 \rightarrow a$$
$$B_1 \rightarrow b$$
}

Here $V_1 = \{S_1, A_1, B_1\}$ and S_1 is a start symbol.

Similarly, we can write $G_2 = \{V_2, \Sigma, P_2, S_2\}$. Here $V_2 = \{S_2, A_2, B_2\}$ with S_2 as start symbol. The P_2 can be

$P_2 = \{$
$$S_2 \rightarrow aA_2A_2 | bB_2B_2$$
$$A_2 \rightarrow b$$
$$B_2 \rightarrow a$$
}

Now let, $L = L_1 \cup L_2$ gives $G \in L$. The G can be written as

$$G = \{V, \Sigma, P, S\}$$
$$V = \{S_1, S_2, A_1, A_2, B_1, B_2\}$$
$$P = P_1 \cup P_2$$

Here S is a start symbol. Then

$P = \{$
$$S \rightarrow S_1 | S_2$$
$$S_1 \rightarrow A_1 S_1 A_1 | B_1 S B_1 | \varepsilon$$
$$A_1 \rightarrow a$$
$$B_1 \rightarrow b$$
$$S_2 \rightarrow aA_2A_2 | bB_2B_2$$
$$A_2 \rightarrow b$$
$$B_2 \rightarrow a$$
}

Thus we obtain context free grammar for $L_1 \cup L_2$.

Step 2 : The context free language is not closed under intersection.

Proof : Consider CFL for $L_1 = \{0^n 1^n 2^i | n \geq 1, i \geq 1\}$ and CFL for $L_2 = \{0^n 1^n 2^n | n \geq 1\}$. If we find $L_1 \cap L_2$ then we obtain a language which can be context free or not context free. That means $L_1 \cap L_2$ is not regular. Hence we can say that CFL is not closed under intersection.

Review Question

1. *What are the closure properties of CFL ? State the proof for any two properties.*

AU : Dec.-10, Marks 8, May-12, Marks 6, May-13, Marks 8

More Solved Examples

Example 4.1 *Construct a PDA accepting the language-*

$\{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}$

Solution : We will first write down the production rules of the CFG for the given language.

$S \rightarrow a\,ABC$

$A \rightarrow a\,A \mid \varepsilon$

$B \rightarrow bB \mid b$

$C \rightarrow cC \mid c$

The given grammar is in GNF. Hence we will write the ID for required PDA as follows.

1) $\delta(q_1, a, S) = (q_1, ABC)$
2) $\delta(q_1, a, A) = (q_1, A)$
3) $\delta(q_1, \varepsilon, A) = (q_1, \varepsilon)$
4) $\delta(q_1, b, B) = (q_1, B)$
5) $\delta(q_1, b, B) = (q_1, \varepsilon)$
6) $\delta(q_1, c, C) = (q_1, C)$
7) $\delta(q_1, c, C) = (q_1, \varepsilon)$
8) $\delta(q_1, \varepsilon, Z_0) = \delta(q_1, \varepsilon) \rightarrow$ ACCEPT state.

Simulation

Now we will simulate the above PDA for the string

$w = abbccc$

$\delta(q_1, abbccc, S) \vdash \delta(q_1, bbccc, ABC)$ ...Rule 1 applied

$\vdash \delta(q_1, bbccc, BC)$ ... Rule 3 applied

$\vdash \delta(q_1, bccc, BC)$ ... Rule 4 applied

$\vdash \delta(q_1, ccc, C)$ ... Rule 5 applied

$\vdash \delta(q_1, cc, C)$ ... Rule 6 applied

$\vdash \delta(q_1, c, C)$ ... Rule 6 applied

$\vdash \delta(q_1, \varepsilon, Z_0)$ ... Rule 7 applied

$\vdash (q_1, \varepsilon)$ ... **ACCEPT state.**

Example 4.2 *Define pushdown automata. Design a PDA for the following language :*

$L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$

Solution : Pushdown automata : Refer section 4.1.

Let, $L = \{a^i b^j c^k \mid i=j \text{ or } j=k\}$ be the given language. We will first write a CFG for the language L. The production rules for this CFG would be -

$S \rightarrow S1 \mid S2$

$S1 \rightarrow a\,AbZ$

$A \rightarrow a\,Ab \mid \varepsilon$

$Z \rightarrow cZ \mid c$

$S2 \rightarrow a\,PbQc$

$P \rightarrow aP \mid \varepsilon$

$Q \rightarrow bQc \mid bc$

Now we will design PDA for this grammar, but we need to convert this grammar to GNF form.

Let,

$S1 \rightarrow a\,AbZ$	$\Rightarrow S1 \rightarrow aABZ$ $B \rightarrow b$
$A \rightarrow a\,Ab$	$\Rightarrow A \rightarrow aAB$
$A \rightarrow \varepsilon$	$\Rightarrow A \rightarrow \varepsilon$
$Z \rightarrow cZ$	Already in GNF
$Z \rightarrow c$	Already in GNF
$S2 \rightarrow a\,PbQc$	$\Rightarrow S2 \rightarrow aPBQT$ $T \rightarrow c$
$P \rightarrow aP \mid \varepsilon$	Already in GNF
$Q \rightarrow bQc$	$\Rightarrow Q \rightarrow bQT$
$Q \rightarrow bc$	$\Rightarrow Q \rightarrow bT$

Hence to summarize,

$S \rightarrow S1 \mid S2$

$S1 \rightarrow a\,ABZ$

$B \rightarrow b$

$A \rightarrow a\,AB$

$A \rightarrow \varepsilon$

$Z \rightarrow cZ$

$Z \rightarrow c$

$S2 \rightarrow aPBQT$

$T \rightarrow c$

$P \rightarrow aP \mid \varepsilon$

$Q \rightarrow bQT$

$Q \rightarrow bT$

The PDA will be -

1) $\delta(q,\varepsilon,S) = (q,S1)$
2) $\delta(q,\varepsilon,S) = (q,S2)$
3) $\delta(q,a,S1) = (q,ABZ)$
4) $\delta(q,b,B) = (q,\varepsilon)$
5) $\delta(q,a,A) = (q,AB)$
6) $\delta(q,\varepsilon,A) = (q,\varepsilon)$
7) $\delta(q,c,Z) = (q,Z)$
8) $\delta(q,c,Z) = (q,\varepsilon)$
9) $\delta(q,a,S2) = (q,PBQT)$
10) $\delta(q,c,T) = (q,\varepsilon)$
11) $\delta(q,a,P) = (q,\varepsilon)$
12) $\delta(q,\varepsilon,P) = (q,\varepsilon)$
13) $\delta(q,b,Q) = (q,T)$
14) $\delta(q,\varepsilon,Z_0) = (q,\varepsilon)$

To simulate ***aabbc***

$\delta(q,\varepsilon,S) = (q,aabbc,S1)$ ∵ Rule 1

$\delta(q, aabbc, S1) \vdash \delta(q,abbc, ABZ)$ ∵ Rule 3

$\vdash \delta(q,abbc, aABBZ)$

$\vdash \delta(q,bbc,ABBZ)$ ∵ Rule 6

$\vdash \delta(q,bbc,BBZ)$ ∵ Rule 4

$\vdash \delta(q,bbc,bBZ)$

$\vdash \delta(q,bc,BZ)$ ∵ Rule 4

$\vdash \delta(q, bc, bZ)$

$\vdash \delta(q, c, Z)$ ∵ Rul

$\vdash \delta(q, c, c)$

$\vdash \delta(q, \varepsilon, Z_0)$ ∵ Rule

$\vdash (q, \varepsilon)$ Acc

Example 4.3 *Consider the given PDA :*

PDA $M = (\{q_0\}, \{0,1\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \phi)$

where δ *is defined as follows :*

$\delta(q_0, 0, Z_0) = \{(q_0, aZ_0)\}$

$\delta(q_0, 1, Z_0) = \{(q_0, bZ_0)\}$

$\delta(q_0, 0, a) = \{(q_0, aa)\}$

$\delta(q_0, 1, b) = \{(q_0, bb)\}$

$\delta(q_0, 0, b) = \{(q_0, \varepsilon)\}$

$\delta(q_0, 1, a) = \{(q_0, \varepsilon)\}$

$\delta(q_0, \varepsilon, Z_0) = \{(q_0, \varepsilon)\}$

Convert the given PDA M to the corresponding CFG.

Solution : Let, the CFG G = (V, T, P, S) for given PDA. The V represents the se non-terminals. These are

$$V = \{S, [q_0 Z_0 q_0], [q_0 a q_0], [q_0 b q_0]\}$$

$$T = \{0, 1\}$$

Now we will consider each δ function from given PDA. Then we will map the (accordingly.

The start symbol S can be -

$$S \rightarrow [q_0 Z_0 q_0]$$

Consider $\delta(q_0, 0, Z_0) = \{(q_0, aZ_0)\}$. The production rule will be -

1. $[q_0 Z_0 q_0] \rightarrow 0\,[q_0 a q_0]\,[q_0 Z_0 q_0]$

Consider $\delta(q_0, 1, Z_0) = \{(q_0, bZ_0)\}$

2. $[q_0 Z_0 q_0] \rightarrow 1\,[q_0 b q_0]\,[q_0 Z_0 q_0]$

Consider $\delta(q_0, 0, a) = \{(q_0, aa)\}$

3. $[q_0 a q_0] \rightarrow 0\,[q_0 a q_0]\,[q_0 a q_0]$

Consider $\delta(q_0, 1, b) = \{(q_0, bb)\}$

4. $[q_0 b q_0] \rightarrow 1\,[q_0 b q_0][q_0 b q_0]$

Consider $\delta(q_0, 0, b) = \{(q_0, \varepsilon)\}$

5. $[q_0 b q_0] \rightarrow 0$

Consider $\delta(q_0, 1, a) = \{(q_0, \varepsilon)\}$

6. $[q_0 a q_0] \rightarrow 1$

Consider $\delta(q_0, \varepsilon, Z_0) = \{(q_0, \varepsilon)\}$

7. $[q_0 Z_0 q_0] \rightarrow \varepsilon$

Example 4.4 *Convert the grammar* $S \rightarrow aSb \mid A, A \rightarrow bSa \mid S \mid \varepsilon$ *to PDA that accepts the same language by empty stack.*

AU : Dec.-11, Marks 10

Solution : We will first simplify the given grammar by eliminating unit and useless productions and then by eliminating ε.

$$S \rightarrow aSb \mid bSa \mid \varepsilon$$

then $$S \rightarrow aSb \mid bSa \mid ab \mid ba.$$

we will now convert this Grammar to GNF

Let,

$$S \rightarrow aSb \Rightarrow S \rightarrow aSB$$
$$B \rightarrow b$$

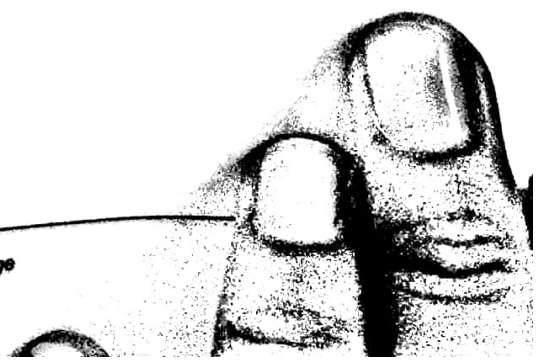
$$S \rightarrow baS \Rightarrow S \rightarrow bSA$$
$$A \rightarrow a$$

$$S \rightarrow ab \Rightarrow S \rightarrow aB$$

$$S \rightarrow ba \Rightarrow S \rightarrow bA.$$

The ID for PDA is

$$\delta(q_0, a, S) = (q_0, SB)$$
$$\delta(q_0, b, S) = (q_0, SA)$$
$$\delta(q_0, a, S) = (q_0, B)$$
$$\delta(q_0, b, S) = (q_0, A)$$
$$\delta(q_0, a, A) = (q_0, \varepsilon)$$
$$\delta(q_0, b, B) = (q_0, \varepsilon)$$



4.9 Two Marks Questions with Answers

Q.1 Define pushdown automata. AU : Dec.-03; May-10

Ans. : The PDA can be defined as a collection of seven components.

1. The finite set of states Q.
2. The input set Σ.
3. Γ is a stack alphabet.
4. q_0 is initial stage, $q_0 \in Q$.
5. Z_0 is a start symbol which is in Γ.
6. Set of final states $F \subseteq Q$.
7. δ is mapping function used for moving from current state to next state.

Q.2 What are the different types of languages accepted by a pushdown automaton and define them ? AU : Dec.-03, 06, 12, May-05

Ans. : PDA accepts two types of languages one is "accepted by final state" and other is "accepted by empty stack".

i) Accepted by Final state – Let

$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be PDA. Then L(P) the language accepted by P by final state is

$\{w \mid (q_0, w, Z_0) \overset{*}{} (q, \varepsilon, \alpha)\}$

{ α is a stack string. That means starting from q_0 with input w and waiting for the input P such that P consumes w from input and enters in accepting state. The stack may or may not be empty.

ii) Accepted by Empty Stack – For each PDA

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$$

$$N(P) = \{w \mid q_0, w, Z_0) \overset{*}{} (q, \varepsilon, \varepsilon)\}$$

where q_0 is the initial state with input w and N(P) represents an empty stack.

Q.3 Give an example of PDA. [Refer example 4.1.1] AU : May-04, Dec.-05

Q.4 Define acceptance of a PDA by empty stack. Is it true that the language accepted by a PDA empty stack or by that of final states are different languages ? AU : May-04, Dec.-05

Ans. : The acceptance of PDA by empty stack can be defined as -

$$N(P) = \{w \mid q_0, w, Z_0) \overset{*}{\vdash} (q_0, \varepsilon, \varepsilon)\}$$