6

Pushdown Automata

Objectives :

- Concept of Pushdown Automata.
- Instantaneous Description.
- Relationship between CFG and PDA.
- CFG to PDA Conversion.
- PDA to CFG Conversion.
- PDA with Two Stacks.

6.1 Introduction

As we have seen in earlier chapters that finite automata is a machine which accepts or models the regular languages.

That means for any regular expression the pictorial representation could be finite automata. By same way as, we have learnt the context free language, we may think as there should be something which can model the CFL. The basic relationship between context free grammar and regular expression is that the CFG can be constructed for every regular expression. But more than that CFG can also be written for non regular languages like $0^n\ 1^n$. Thus regular languages are subset of context free languages.

It will always be effective if we can model this context free language. If we think of FA to model any CFG then it shows its limitations. In other words FA can not model some languages like $0^n\ 1^n$. This is a non regular language. So we can declare one important property and that is : For every regular expression the finite automata can be drawn, whereas we can not draw finite automata for any non regular language. Similarly FA is not sufficient to model any context free language. So a new model has come up, which is called pushdown automata. The speciality of this machine is that it has a stack which is used to remember the input. Thus in PDA memory is used by means of stack.

In this chapter we will learn the concept of pushdown Automata with the help of definition, model and some interesting examples. Then we will discuss the relationship of PDA and CFL.

6.2 Definition of Pushdown Automata

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.

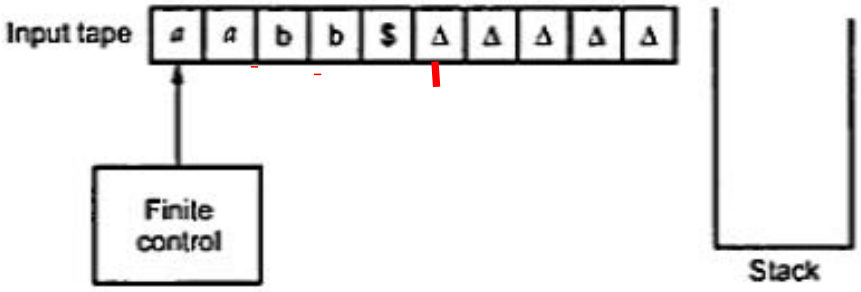


Fig. 6.1 Pushdown automata

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.

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.

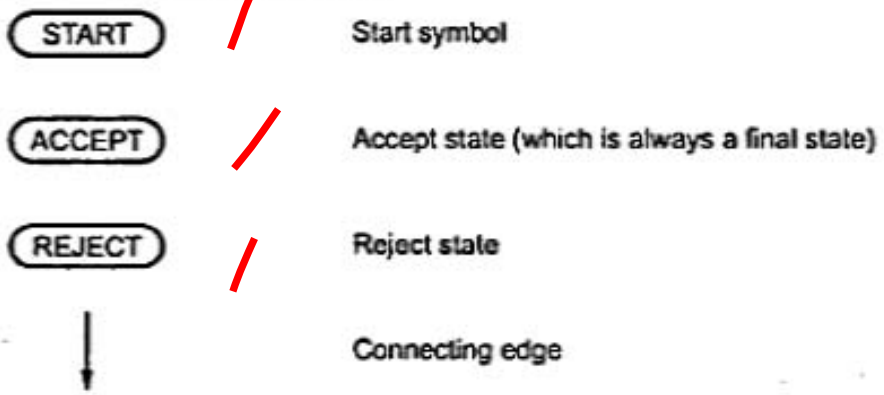


Fig. 6.2 Symbols used in PDA

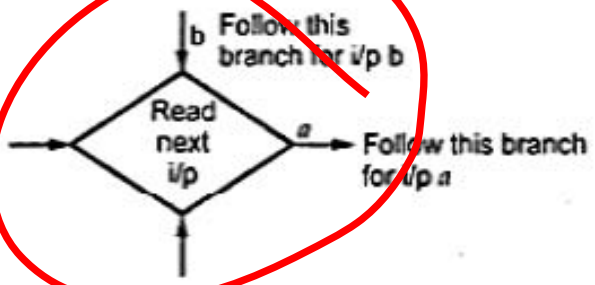


Fig. 6.3 Read symbol

Example 6.1 : *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 -

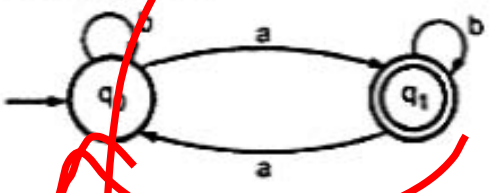


Fig. 6.4

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 -

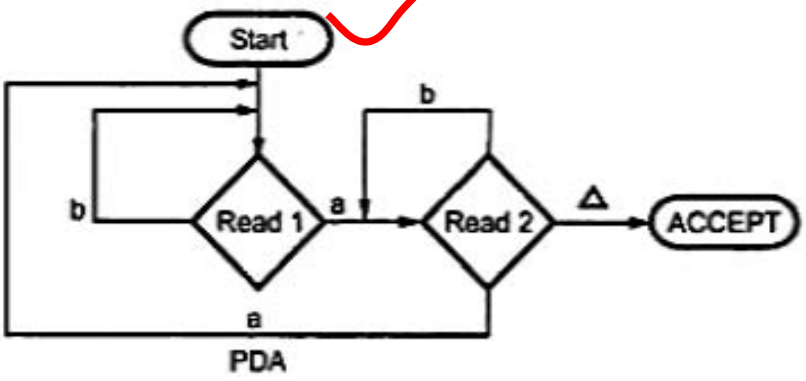


Fig. 6.5 PDA

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.

b	a	a	a	b	Δ	Δ	Δ

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 6.2 : *Design PDA for the language L = {001}.*

Solution : The simple FA for this language is

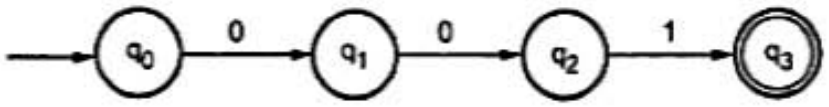


Fig. 6.6

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

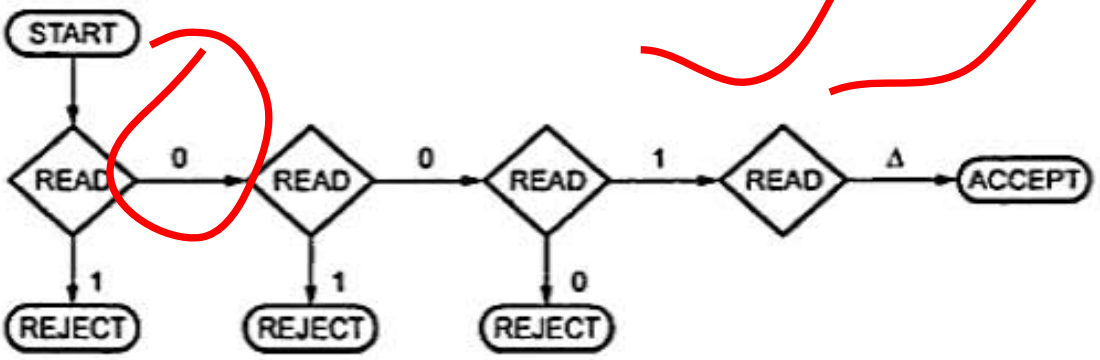


Fig. 6.7

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.

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

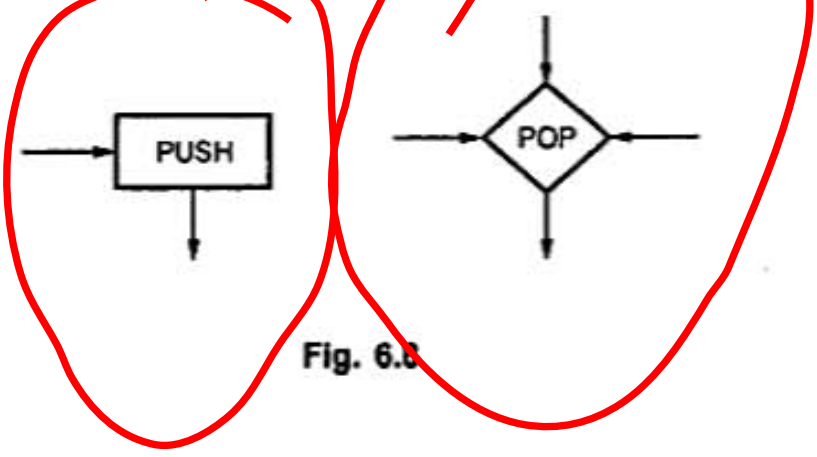


Fig. 6.8

Example 6.3 : *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 *aaaa*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.

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

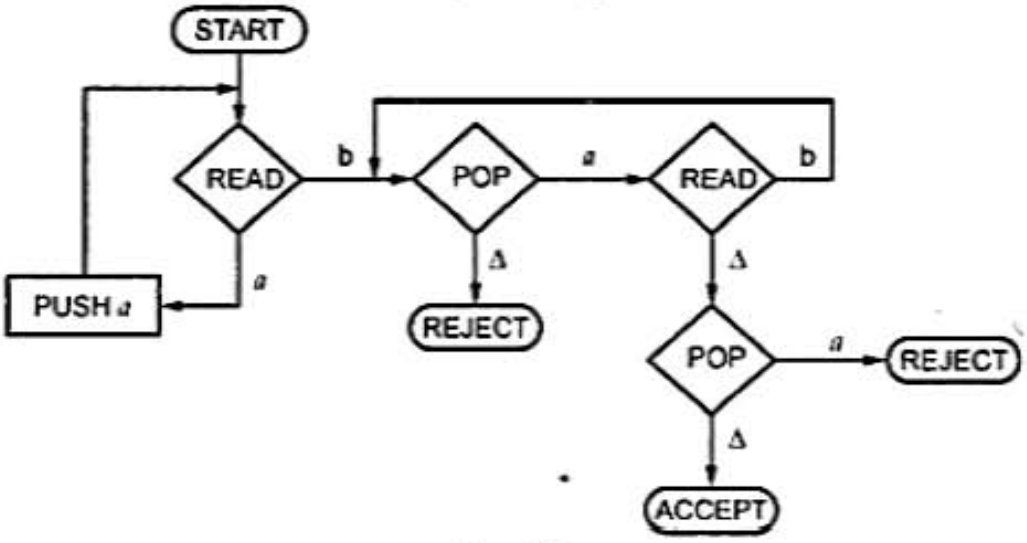


Fig. 6.9

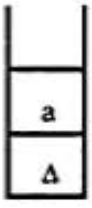
The simple logic we have applied here is that we are starting with start symbol. At the first read statement if we read *a*, we push that *a* onto the stack. This will be repeated by applying a self loop to this read statement. By this all the *a*'s will be pushed onto the stack. Now when we will read 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.

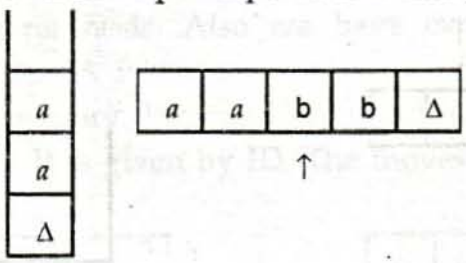
Step 1 : We will start with start symbol.

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

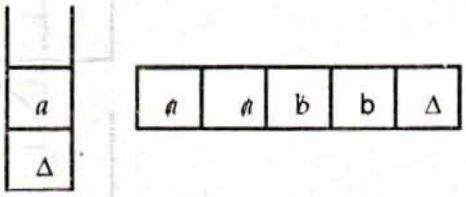


e stack initially, contains Δ that means it is empty.

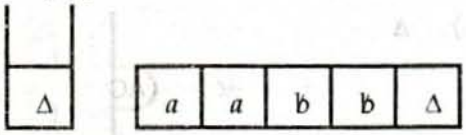
ep 3 : Read next symbol from input tape. It is *a* and push it onto the stack.



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



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



ote that both tape and stack both are empty.

Example 6.4 : *Design PDA that checks the well formedness of parenthesis.*

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

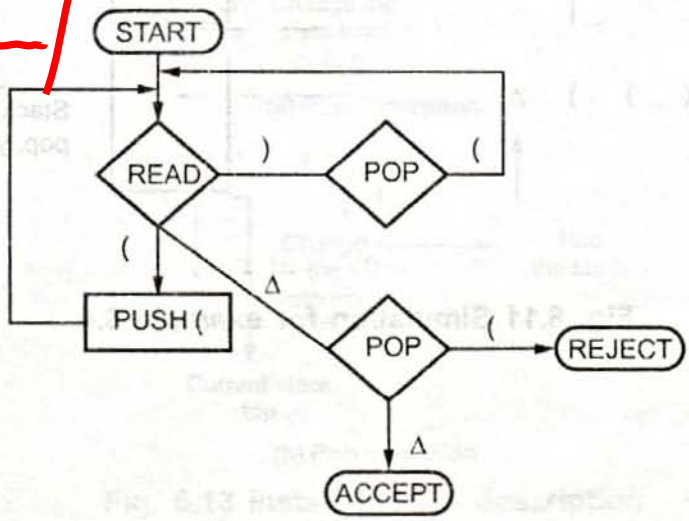


Fig. 6.10 Well formedness of parenthesis

Let us simulate this PDA for the input
Hence we reach to accept state.

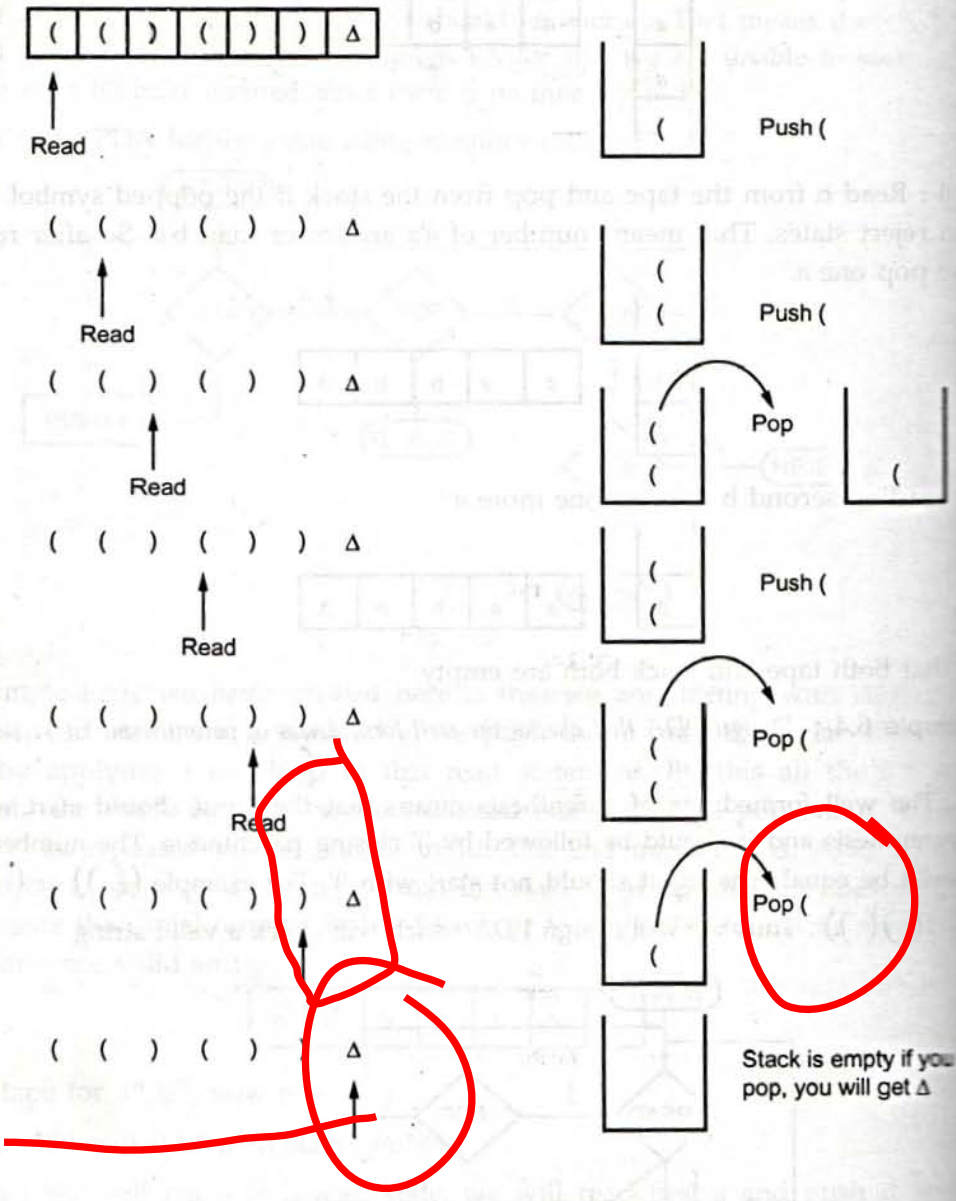


Fig. 6.11 Simulation for example 6.4

6.3 Instantaneous Description

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.

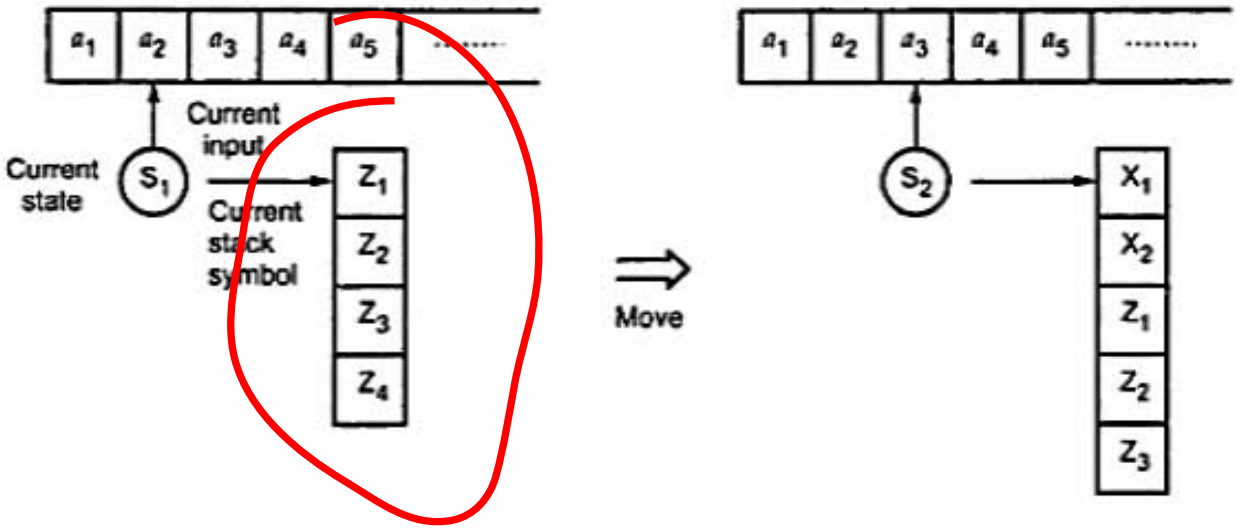


Fig. 6.12

From above Fig. 6.12, 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 -

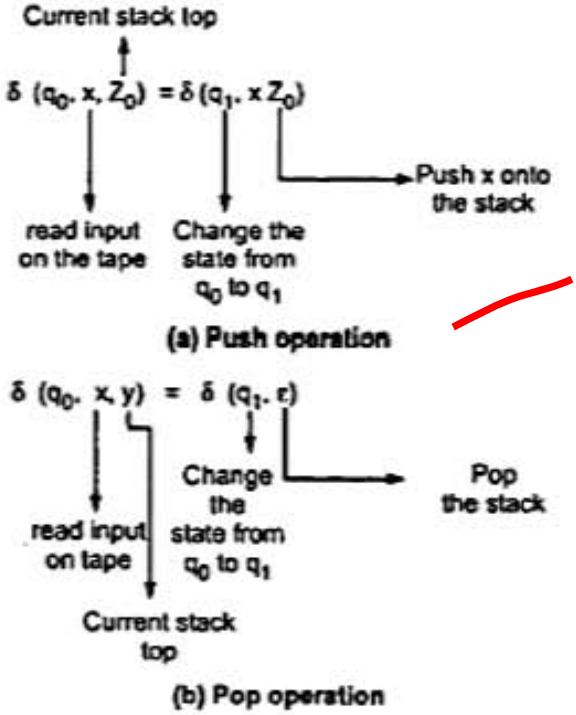


Fig. 6.13 Instantaneous description

Now let us solve some example based on it.

Example 6.5 : *Design a PDA for accepting a language* $\{L = a^n b^n \mid n \geq 1\}$.

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's 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 -

$$\begin{aligned}(q_0, aaabbb, Z_0) &\vdash (q_0, aabbb, aZ_0)\\ &\vdash (q_0, abbb, aaZ_0)\\ &\vdash (q_0, bbb, aaaZ_0)\\ &\vdash (q_1, bb, aaZ_0)\\ &\vdash (q_1, b, aZ_0)\\ &\vdash (q_1, \varepsilon, Z_0)\\ &\vdash (q_2, \varepsilon)\end{aligned}$$

ACCEPT state.

Example 6.6 : *Construct PDA for the language* $L = \{a^n b^{2n} \mid n \geq 1\}$

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 be popped from the stack. This basically maintains the $a^n b^{2n}$ count and sequence. The PDA 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.

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

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"

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

$\vdash (q_0, abbbbbb, aaaaZ_0)$

$\vdash (q_0, bbbbbb, aaaaaaZ_0)$

$\vdash (q_1, bbbbb, aaaaaZ_0)$

$\vdash (q_1, bbbb, aaaaZ_0)$

$\vdash (q_1, bbb, aaaZ_0)$

$\vdash (q_1, bb, aaZ_0)$

$\vdash (q_1, b, aZ_0)$

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

$\vdash (q_2, \varepsilon)$

Final state or ACCEPT state.

Thus input gets accepted by using the constructed PDA.

Example 6.7 : *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.

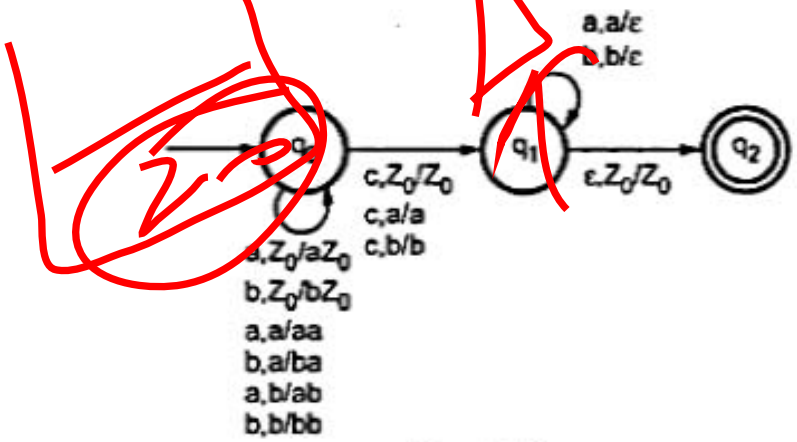


Fig. 6.14

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.

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 6.8 : *Design PDA to accept the language*

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

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 6.9 : *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

$$\begin{aligned}
\delta(q_0, aababab, Z_0) &\vdash (q_0, ababab, aZ_0) \\
&\vdash (q_0, babab, aZ_0) \\
&\vdash (q_0, abab, aZ_0) \\
&\vdash (q_0, bab, aaZ_0) \\
&\vdash (q_0, ab, aZ_0) \\
&\vdash (q_0, b, aaZ_0) \\
&\vdash (q_0, \varepsilon, aZ_0) \\
&\vdash (q_f, a) \\
&\quad \text{Accept state.}
\end{aligned}$$

Thus input gets accepted.

Example 6.10 : *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 -

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

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

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

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

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

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

$$\delta(q_0, \varepsilon, b) = (q_f, b) \leftarrow \text{The b's are more in number.}$$

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

$$\begin{aligned}
\delta(q_0, abbab, Z_0) &\vdash (q_0, bbab, aZ_0) \\
&\vdash (q_0, bab, Z_0) \\
&\vdash (q_0, ab, bZ_0) \\
&\vdash (q_0, b, Z_0) \\
&\vdash (q_0, \varepsilon, bZ_0) \\
&\vdash (q_f, b)
\end{aligned}$$

Accept state.

Example 6.11 : *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.

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

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

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

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 6.12 :** *Construct PDA for the language* $\{L = a^m b^m c^n \mid m, n \geq 1\}$

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.

$\delta(q_0, aabbccc, Z_0) \vdash \delta(q_0, abbccc, aZ_0)$

$\vdash \delta(q_0, bbccc, aaZ_0)$

$\vdash \delta(q_1, bccc, aZ_0)$

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

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

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

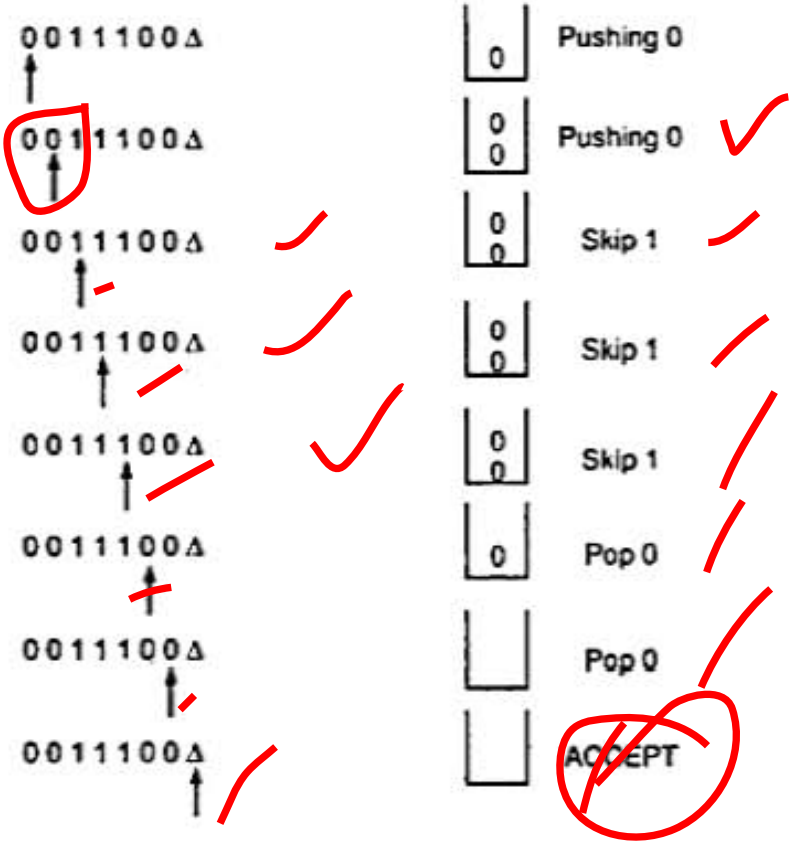
$\vdash \delta(q_1, \varepsilon, Z_0)$

$\vdash \delta(q_2, Z_0)$

ACCEPT

Example 6.13 : *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

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

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

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

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

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

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

For example :

$$\begin{aligned}
\delta(q_0, 0011100, Z_0) &\vdash \delta(q_0, 011100, 0Z_0) \\
&\vdash \delta(q_0, 11100, 00Z_0) \\
&\vdash \delta(q_0, 1100, 00Z_0) \\
&\vdash \delta(q_1, 100, 00Z_0) \\
&\vdash \delta(q_1, 00, 00Z_0) \\
&\vdash \delta(q_1, 0, 0Z_0) \\
&\vdash \delta(q_1, \varepsilon, Z_0) \\
&\vdash \delta(q_2, Z_0) \\
&\quad \text{ACCEPT}
\end{aligned}$$

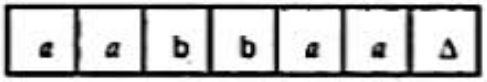
6.4 Non Deterministic Pushdown Automata

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 6.14 :** *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, *bab*, *bbabb*, *abba* .... }. 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.

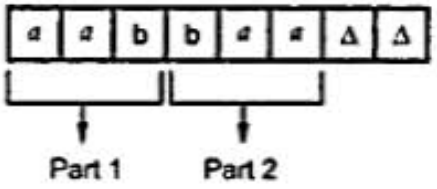


Fig. 6.15

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

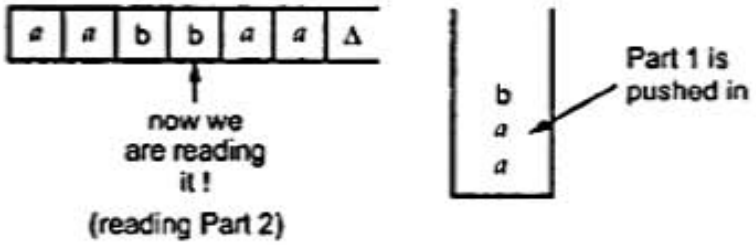


Fig. 6.16

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.

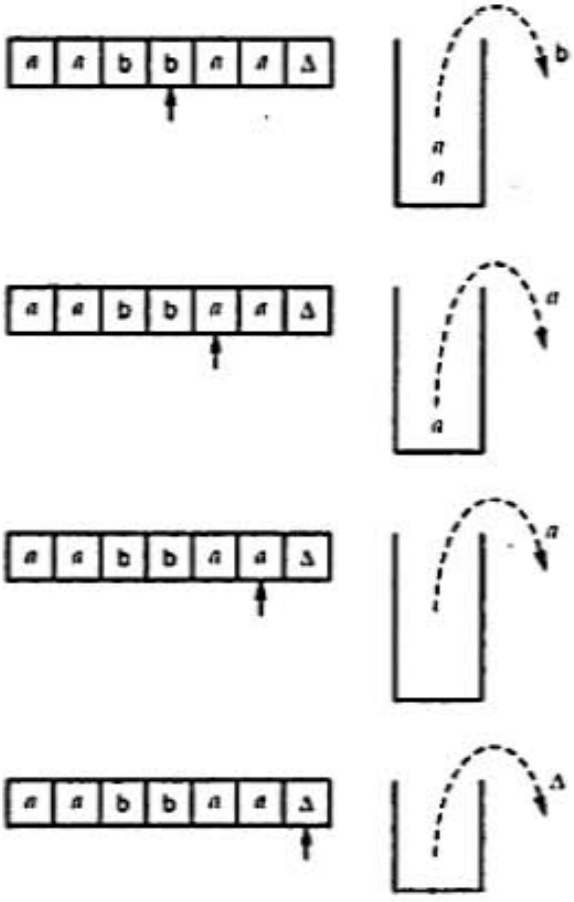


Fig. 6.17

The NPDA is as below -

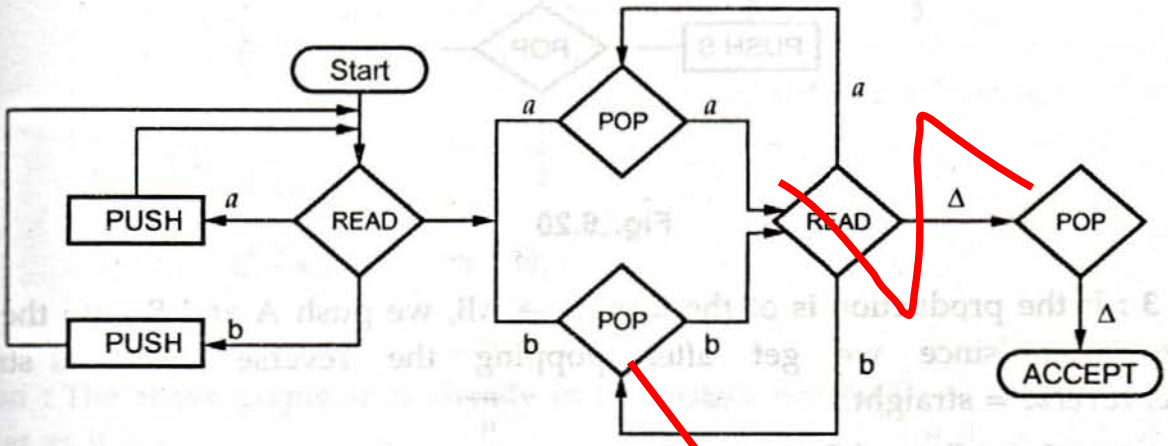


Fig. 6.18

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 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}.

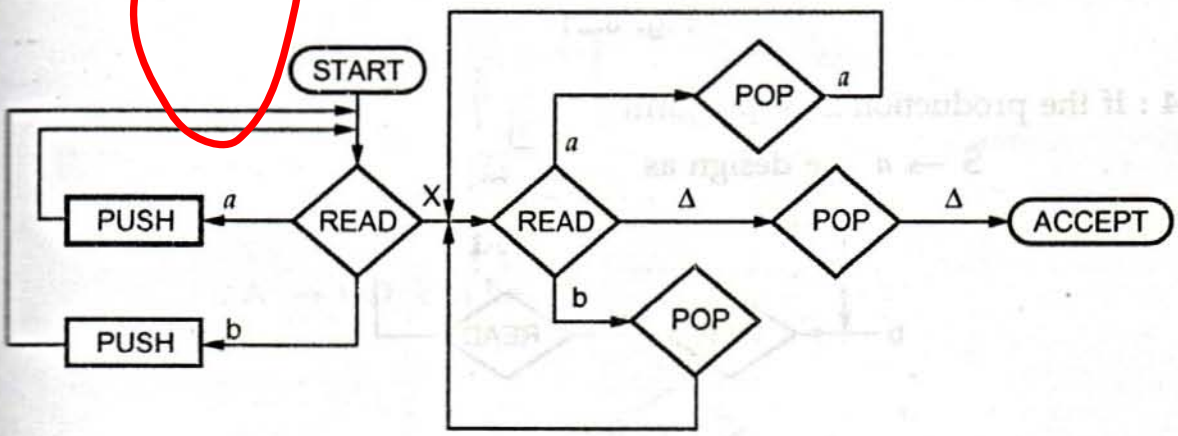


Fig. 6.19

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

6.4.1 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 other production rules for the start symbol, we immediately perform the pop operation. It is as shown below.

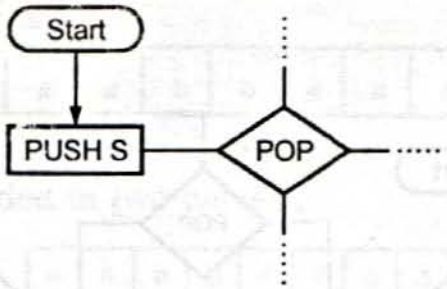


Fig. 6.20

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 $S \rightarrow AB$

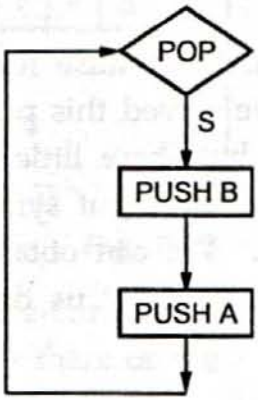


Fig. 6.21

Step 4 : If the production is of the form

$S \rightarrow a$ we design as

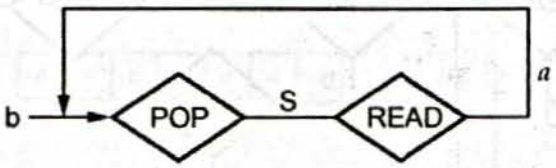


Fig. 6.22

This means replacing a nonterminal s by *a*.

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. 6.23

Example 6.15 : *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

$S \rightarrow AB$

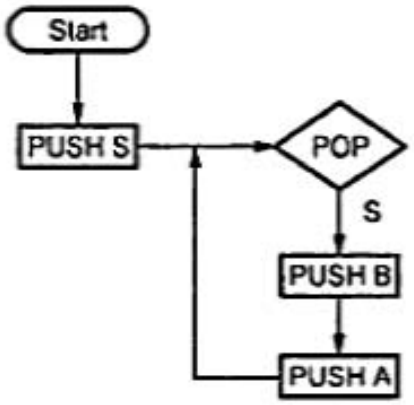


Fig. 6.24

$A \rightarrow CD$ can be

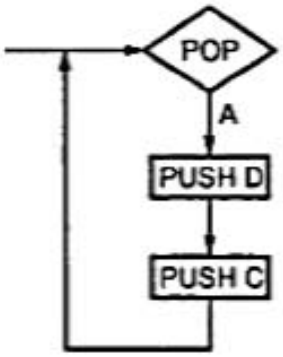


Fig. 6.25

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

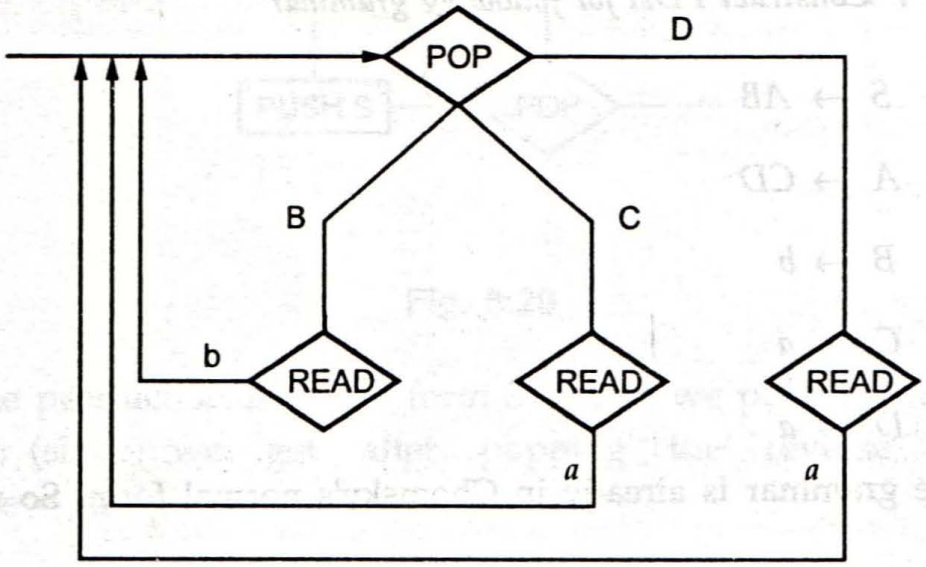


Fig. 6.26

Finally we could draw the complete PDA as

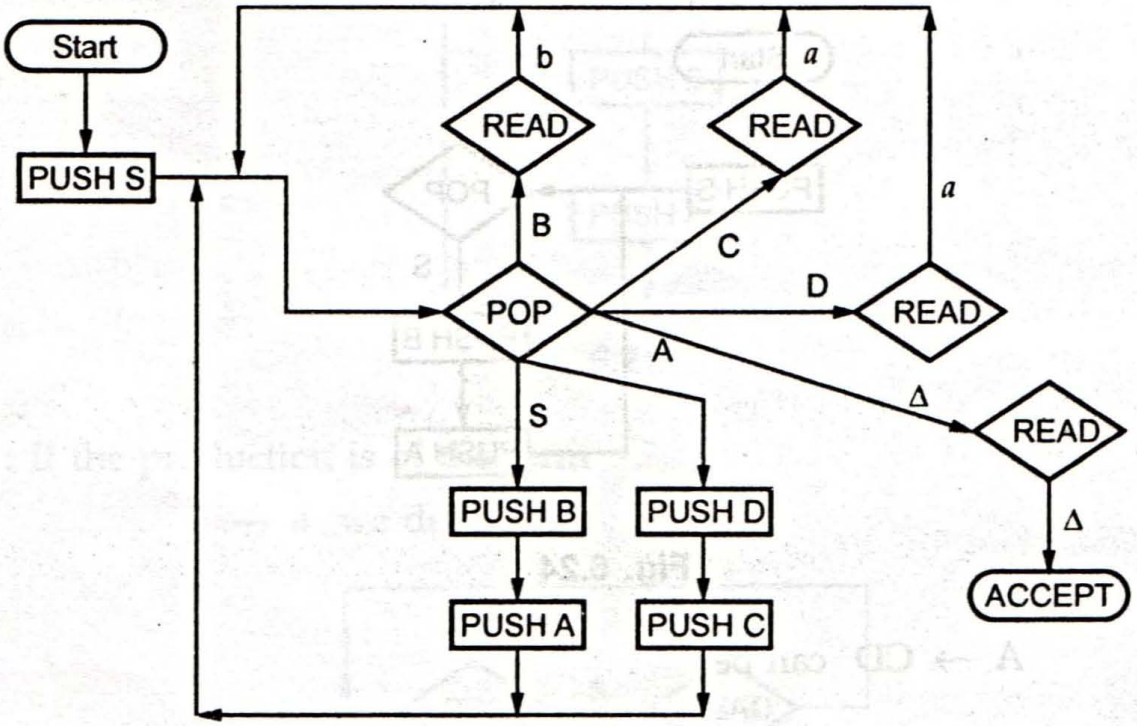


Fig. 6.27

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

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

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

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

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

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

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 6.16 : *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

$$S \rightarrow AS$$

$$S \rightarrow BS$$

$$A \rightarrow 0$$

$$B \rightarrow 1$$

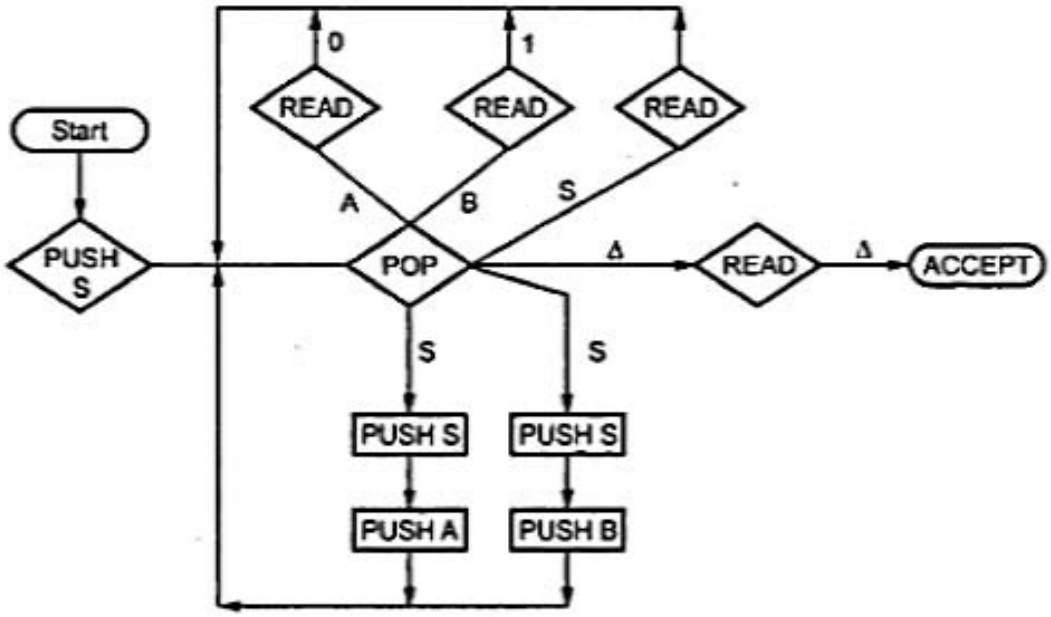


Fig. 6.28

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.

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,

$$\begin{aligned}
\delta (q, 0101, S) &\vdash (q, \underline{0}101, \underline{A}\, S) \\
&\vdash (q, 101, S) \\
&\vdash (q, \underline{1}01, \underline{B}\, S) \\
&\vdash (q, 01, S) \\
&\vdash (q, \underline{0}1, \underline{A}\, S) \\
&\vdash (q, 1, S) \\
&\vdash (q, 1, BS) \\
&\vdash (q, \varepsilon, S) \\
&\vdash (q, \varepsilon)
\end{aligned}$$

Accepting configuration.

➡ **Example 6.17 :** *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 -

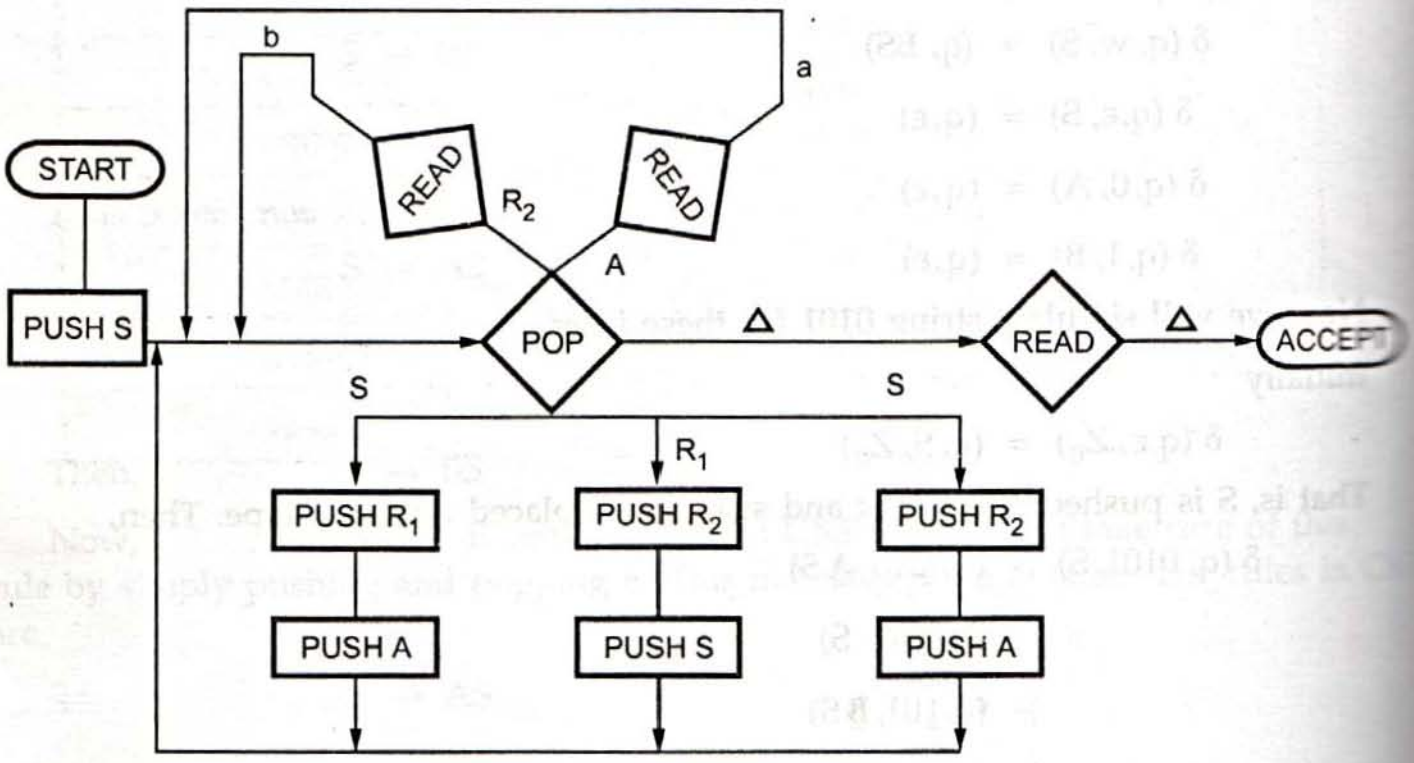


Fig. 6.29

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	$R_2\,\Delta$	Push S
abbΔ	$S\,R_2\,\Delta$	POP
abbΔ	$R_2\,\Delta$	Push R_2
abbΔ	$R_2R_2\,\Delta$	Push A
abbΔ	$A\,R_2R_2\,\Delta$	POP
abbΔ	$R_2R_2\,\Delta$	Read a
bbΔ	$R_2R_2\,\Delta$	POP
bbΔ	$R_2\,\Delta$	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.

6.4.2 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 6.18 : *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.

Example 6.19 : *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. $(q_0\,Z\,q_1) \rightarrow 0(q_0\,A\,q_0)(q_0\,Z\,q_0) \mid 0(q_0\,A\,q_1)(q_1\,Z\,q_0)$

$\delta(q_0, 0, A) = (q_0, AA)$ ... 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. $(q_0\,A\,q_1) \rightarrow 0(q_0\,A\,q_0)(q_0\,A\,q_0) \mid 0(q_0\,A\,q_1)(q_1\,A\,q_1)$

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

6.5 PDA with Two Stacks

There are certain languages which can not be accepted by push down automata as there is only one stack. This is specially when we want to perform two checks on the input string. But if we use two stacks there then those languages can be accepted by PDA. For instance consider a language $L = \{a^n b^n c^n \mid n \geq 1\}$. Now if we use one stack

here then the language can not be accepted by PDA. Suppose with one stack we will push all a's onto the stack on reading them. Then on reading b's we will pop each a. This helps in making a check on equal number of a's and b's. But now when we read out c's the single stack is empty and we can not make out the equality of a's and b's with c's. Thus it is not possible for us to design a PDA with one stack which accepts language $L = a^n b^n c^n$. But if we use two stacks one for storing all a's. Other for storing all the b's then on reading each c we can pop each a from first stack and each b from second stack. This helps in identifying whether or not there are equal number of a's, b's and c's coming in order. This also means that PDA with two stack has more power than PDA with one stack.

The PDA with two stacks accepting $a^n b^n c^n$ is as given below.

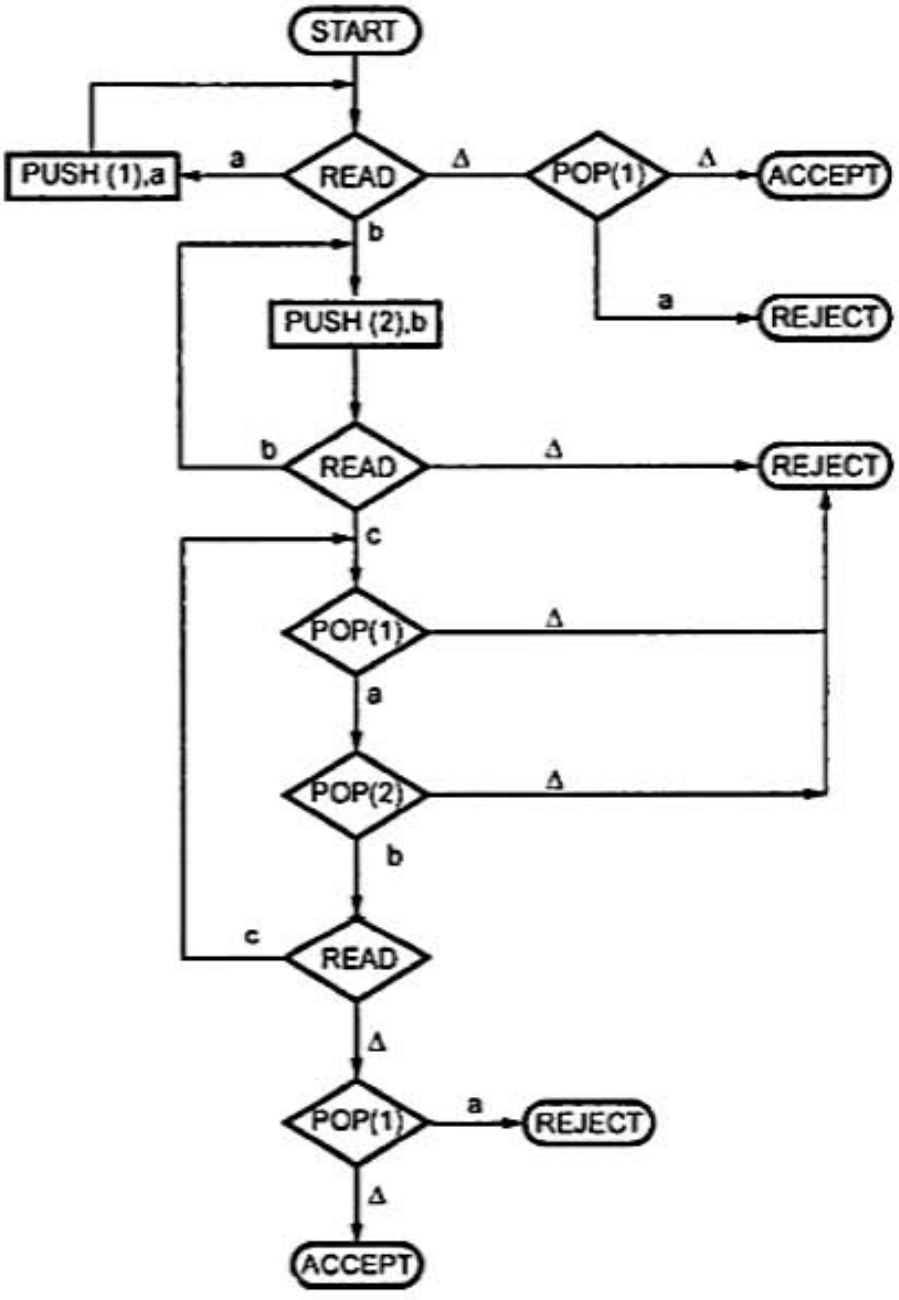


Fig. 6.30 PDA with two stacks

In the given PDA we have used two stacks. Thus push and pop operations are performed on both the stacks PUSH(1) *'a'* means push symbol *'a'* onto stack 1. Similarly PUSH(2), b means push b onto stack 2. The POP (1) denotes pop operation for stack 1 and POP(2) means pop from stack 2. Thus the language like $\{a^n b^n c^n \mid n \geq 0\}$ is accepted by two stacks and not by one stack.

6.6 Deterministic Pushdown Automata and Deterministic Context Free Languages

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

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

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

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

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

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

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

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

Is the PDA deterministic?

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

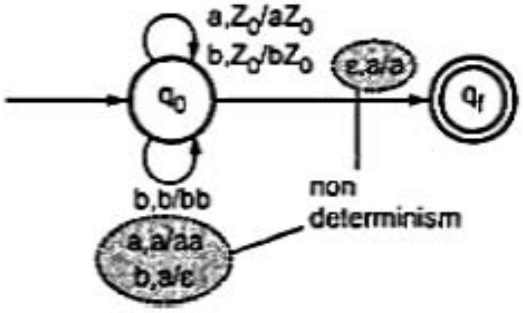


Fig. 6.31

We can convert it into an equivalent DPDA as -

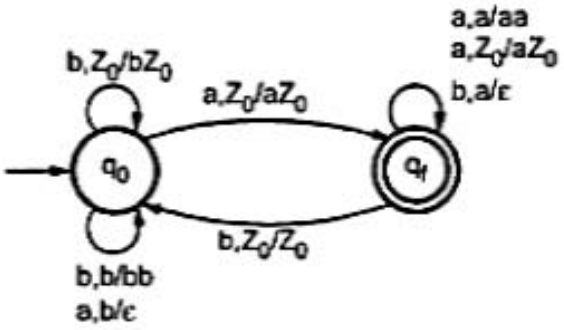


Fig. 6.32

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 6.21 : *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 -

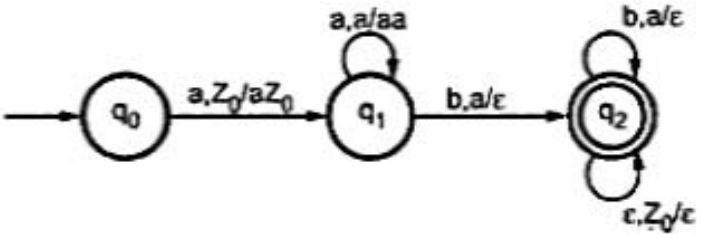


Fig. 6.33

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.

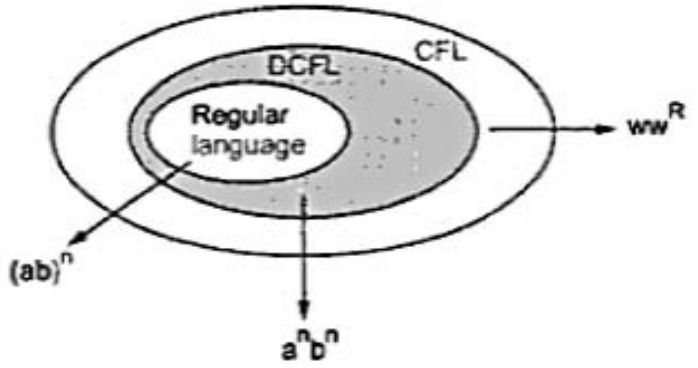


Fig. 6.34

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

- 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.

Review Questions

1. *Construct a PDA for the following language $\{a^m b^n \mid m > n \geq 1\}$.*
2. *Compare Pushdown automata with finite automata.*
3. *Obtain equivalent PDA for the given context free grammar.*

   $S \rightarrow AB$

   $A \rightarrow B$

   $A \rightarrow a$

   $B \rightarrow a$

   $B \rightarrow b$
4. *Is PDA equivalent to NPDA ? Justify.*

❑❑❑