So, the final grammar obtained which is in GNF is shown below:

$A_1 \rightarrow 0A_1A_2 \mid 1A_2 \mid 0A_1Z\,A_2 \mid 1Z\,A_2 \mid 0$
$A_2 \rightarrow 0A_1 \mid 1 \mid 0A_1Z \mid 1Z$
$Z \rightarrow 0A_1A_1 \mid 1A_1 \mid 0A_1Z\,A_1 \mid 1ZA_1$
$Z \rightarrow 0A_1A_1Z \mid 1A_1Z \mid 0A_1ZA_1Z \mid 1ZA_1Z$

Exercise:

1. Is the following grammar ambiguous ?
   $S \rightarrow aSb \mid SS \mid \varepsilon$
2. Show that the following grammar is ambiguous.
   $S \rightarrow SbS \mid a$
3. Let G = (V, T, P, S) be a CFG, where
   V = {E, I}
   T = {a, b, c, d, +, *,(,) }
   P = {
   $E \rightarrow I \mid E+E \mid E*E \mid (E)$
   $I \rightarrow a \mid b \mid c \mid d$
   }
   Obtain the derivations for the strings (a+b)*c*d and a+b*c and the parse tree for each derivation.
4. Obtain a reduced grammar for the grammar shown below
   $S \rightarrow aAa$
   $A \rightarrow Sb \mid bCC \mid aDA$
   $C \rightarrow ab \mid aD$
   $E \rightarrow aC$
   $D \rightarrow aAD$
5. Find an equivalent grammar without ε-productions for the grammar shown below.
   $S \rightarrow aSa \mid bSb \mid A$
   $A \rightarrow aBb \mid bBa$
   $B \rightarrow aB \mid bB \mid \varepsilon$
6. Remove the unit productions from the grammar
   $S \rightarrow A \mid B \mid Cc$
   $A \rightarrow aBb \mid B$
   $B \rightarrow aB \mid bb$
   $C \rightarrow Cc \mid B$
7. Eliminate useless productions from the grammar
   $S \rightarrow aA \mid a \mid B \mid C$
   $A \rightarrow aB \mid \varepsilon$
   $B \rightarrow aA$
   $C \rightarrow cCD$
   $D \rightarrow abd$ Note: First remove ε and unit productions, then simplify CFG.

8. Obtain the following grammar in CNF

$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid I$
$I \rightarrow a \mid b \mid c \mid Ia \mid Ib \mid Ic$

9. Obtain the following grammar in CNF and GNF

$S \rightarrow aA \mid a \mid B \mid C$
$A \rightarrow aB \mid \varepsilon$
$B \rightarrow aA$
$C \rightarrow cCD$
$D \rightarrow abd$

Note: Eliminate ε-productions, unit productions and then convert into normal forms.

10. Simplify the following CFG and convert it into CNF

$S \rightarrow AaB \mid aaB$
$A \rightarrow \varepsilon$
$B \rightarrow bbA \mid \varepsilon$

11. Convert the following grammar into GNF

$S \rightarrow abAB$
$A \rightarrow bAB \mid \varepsilon$
$B \rightarrow Aa \mid \varepsilon$

12. Explain the method of substitution with example
13. What is left recursion? How it can be eliminated?
14. Eliminate left recursion from the following grammar

$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$

15. Eliminate the useless symbols in the grammar

$S \rightarrow aA \mid bB$
$A \rightarrow aA \mid a$
$B \rightarrow bB$
$D \rightarrow ab \mid Ea$
$E \rightarrow aC \mid d$

16. Eliminate left recursion from the following grammar

$S \rightarrow Ab \mid a$
$A \rightarrow Ab \mid Sa$

17. What is the need for simplifying a grammar?
18. What is a useless symbol? Explain with example.
19. Simplify the following grammar

$S \rightarrow aA \mid a \mid Bb \mid cC$
$A \rightarrow aB$
$B \rightarrow a \mid Aa$
$C \rightarrow cCD$
$D \rightarrow ddd$

20. What is an ε - production? What is a Nullable variable? Explain with example.

21. Eliminate all ε- productions from the grammar
$$S \rightarrow ABCa \mid bD$$
$$A \rightarrow BC \mid b$$
$$B \rightarrow b \mid \varepsilon$$
$$C \rightarrow c \mid \varepsilon$$
$$D \rightarrow d$$
22. Eliminate all ε- productions from the grammar
$$S \rightarrow BAAB$$
$$A \rightarrow 0A2 \mid 2A0 \mid \varepsilon$$
$$B \rightarrow AB \mid 1B \mid \varepsilon$$
23. What is an unit production? Give general method of eliminating unit productions from the grammar.
24. Eliminate all unit productions from the grammar
$$S \rightarrow AB$$
$$A \rightarrow a$$
$$B \rightarrow C \mid b$$
$$C \rightarrow D$$
$$D \rightarrow E \mid bC$$
$$E \rightarrow d \mid Ab$$
25. What is Chomsky Normal Form (CNF) and Greiback Normal Form(GNF)?
26. Prove that for every CFG we can have an equivalent grammar using CNF notations where a language does not contain ε.
27. Eliminate unit productions from the grammar
$$S \rightarrow A0 \mid B$$
$$B \rightarrow A \mid 11$$
$$A \rightarrow 0 \mid 12 \mid B$$
28. Eliminate unit productions from the grammar
$$S \rightarrow Aa \mid B \mid Ca$$
$$B \rightarrow aB \mid b$$
$$C \rightarrow Db \mid D$$
$$D \rightarrow E \mid d$$
$$E \rightarrow ab$$
29. Consider the grammar in CNF
$$S \rightarrow 0A \mid 1B$$
$$A \rightarrow 0AA \mid 1S \mid 1$$
$$B \rightarrow 1BB \mid 0S \mid 0$$
30. What is the general procedure to convert a grammar into its equivalent GNF notation?
31. Convert the following grammar into GNF.
$$S \rightarrow AB1 \mid 0$$
$$A \rightarrow 00A \mid B$$
$$B \rightarrow 1A1$$

32. Convert the following grammar into GNF

$$A \rightarrow BC$$
$$B \rightarrow CA \mid b$$
$$C \rightarrow AB \mid a$$

Summary:

What we will know after reading this chapter?

- Method of substitution
- Left recursion
- Procedure to eliminate left recursion
- Simplification of the grammar with proof
- Solutions to simplify various types of grammars
- Useless variable
- ε - Production
- Nullable variable
- Elimination of ε - productions
- Unit production
- Elimination of unit productions
- Chomsky Normal Form (CNF)
- Conversion of various types of grammars to CNF notation
- Greiback Normal Form (GNF)
- Conversion of various types of grammars to GNF notations
- Solution to more than 15 problems of various nature

7

Pushdown Automata

What we will know after reading this chapter?

- Difference between finite automaton (FA) and pushdown automaton (PDA)
- Pushdown automaton
- The transition diagram and moves of PDA
- Actions performed by PDA
- Instantaneous description
- Languages accepted by PDA by a final state
- Languages accepted by PDA by an empty stack
- Various ways of constructing PDA for the given languages
- Deterministic and non-deterministic PDA
- To obtain PDA from CFG, the method and solution to various types of problems
- Applications of GNF
- To obtain CFG from PDA
- Solutions to more than 23 problems of various nature

In chapter 5 and 6 we have seen how a context free language can be described using context free grammars and we have in fact defined some of the C programming constructs using BNF notations (see section 5.8). But, we have not encountered an automaton to recognize the context free languages as we had finite automata to recognize the regular languages. A DFA (or NFA) is not powerful enough to recognize many context-free language. Since the finite automaton have strict finite memories, it is not possible to construct those machines to accept the context free languages. The automaton to recognize the CFL may require additional amount of storage which

will be used to store the data. For example, during parentheses matching, if we encounter '(', it is required to store the left parentheses on the stack and as we encounter right parentheses i.e., ')' we may have to pop the corresponding '(' from the stack. Thus, stack is used to remember the scanned symbols. This is evident as we design the automaton to accept CFL. Sometimes, it is required to count the symbols also. For example, if we have the language $L = \{a^n b^n \mid n \geq 0\}$ after reading n number of a's, we should see that n number of b's exist. So, an automaton that we construct to accept CFL's should be in a position to count at the same time should have sufficient memory to hold the string scanned.

Since the DFA's or NFA's can not count and can not store the input for future reference, we are forced us to have a new machine called **Pushdown Automaton (PDA)**. Note that **PDA** is an NFA with the exception that PDA has an extra stack. So, the definition of PDA is similar to the definition of NFA with slight changes. Let us take some typical examples and define the PDA.

Example 7.1: Consider the language $L = \{wCw^R \mid w \in (a + b)^*\}$ where w^R is reserve of w and $C \in \Sigma$ indicates middle of string.

It is clear from the language $L = wCw^R$ that if w is the string abb then w^R is reverse of w i.e., bba. The language generated will be abbCbba which is a palindrome. So, the language generates strings of palindromes.

Let us device a method to accept a string of palindrome. In a given string, the letter C is the middle of the string. To start with, machine will be in the start state say q_0. Now keep pushing all input symbols on to the stack and stay in state q_0 till the input symbol C is encountered.

Immediately after scanning the input symbol C, change the state to q_1. Now, we have passed the middle of the string. If the given string is a palindrome, for each character encountered after the midpoint, there will be a corresponding character on the stack.

So, in state q_1, after scanning the input symbol, compare it with most recently pushed symbol on the stack. If they are same, discard both the symbols and scan the next symbol and compare it with the symbol on the stack and repeat the process and remain in state q_1. If there is a mismatch the machine halts and the string will be rejected. Once the last symbol in the input is matched with symbol on the stack, finally stack will be empty. Once the stack is empty, it is evident that the given string is a palindrome. So, change the state to q_2 which is an accepting state.

Note: It is clear from this example that, the PDA has set of states Q and set of input symbols Σ. Since it has stack also as a component, some symbols will be pushed on to the stack. So, the stack has stack alphabets denoted by Γ. The stack contains a special symbol Z_0 which is present in the stack initially (Note that this is the initial condition). It means that an empty stack contains Z_0 as the top of the stack.

7.1 Transitions

For the example shown in 7.1, the transitions can be defined as shown in table 7.1. Let us not worry about how to get these transitions and it will be discussed later. Assume that q_0 is the start state and Z_0 is the initial symbol on the stack indicating stack is empty.

Since there is an ε-transition, the PDA is actually Non-deterministic. In a non-deterministic PDA, there will be ε-transitions or there may be multiple transitions from the same state on a given input when a specified symbol is there on the stack.

$\delta(q_0, a, Z_0)$	=	(q_0, aZ_0)
$\delta(q_0, b, Z_0)$	=	(q_0, bZ_0)
$\delta(q_0, a, a)$	=	(q_0, aa)
$\delta(q_0, b, a)$	=	(q_0, ba)
$\delta(q_0, a, b)$	=	(q_0, ab)
$\delta(q_0, b, b)$	=	(q_0, bb)
$\delta(q_0, c, Z_0)$	=	(q_1, Z_0)
$\delta(q_0, c, a)$	=	(q_1, a)
$\delta(q_0, c, b)$	=	(q_1, b)
$\delta(q_1, a, a)$	=	(q_1, ε)
$\delta(q_1, b, b)$	=	(q_1, ε)
$\delta(q_1, \varepsilon, Z_0)$	=	(q_2, Z_0)

δ: $Q \times (\Sigma \cup \varepsilon) \times \Gamma$ to $Q \times \Gamma^*$

Table 7.1 Transitions

For example, consider the transitions

$$\delta(q_0, a, Z) = (q_1, bZ)$$
$$\delta(q_0, a, Z) = (q_2, cZ)$$

These transitions can also be written as

$$\delta(q_0, a, Z) = \{ (q_1, bZ), (q_2, cZ) \}$$

In the above transition, when the PDA is currently in state q_0, on scanning the input symbol *a* and when the top of the stack contains Z, the machine can perform one of the transitions defined i.e., either

1. The PDA can go to state q_1 after pushing the symbol *b* on to the stack

   or

2. The PDA can go to state q_2 after pushing the symbol *c* on to the stack

Unless otherwise specified, let us call **Push Down Automata (PDA)** as **Non Deterministic Push Down Automata (NPDA)**. The formal definition of PDA is shown below:

Definition: A pushdown automata (PDA) is a seven tuple

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

where

Q - is set of finite states.

Σ - Set of input alphabets.
Γ - Set of stack alphabets.
δ - transition from $Q \times (\Sigma \cup \varepsilon) \times \Gamma$ to finite sub set of $Q \times \Gamma^*$
δ is called the transition function of M.
$q_0 \in Q$ is the start state of machine.
$Z_0 \in \Gamma$ is the initial symbol on the stack.
$F \subseteq Q$ is set of final states.

The actions (i.e., transitions) performed by the PDA depends on

1. The current state.
2. The next input symbol
3. The symbol on top of the stack.

The action performed by the machine consists of

1. Changing the states from one state to another
2. Replacing the symbol on the stack.

Note: In general, the transition function accepts three parameters namely a state, an input symbol and stack symbol and returns a new state after changing the top of the stack i.e. the transition function has the form:

$$\delta \text{ (state, input\_symbol, stack\_symbol)} = \text{(next\_state, stack\_symbol)}$$

Example 7.2: What does each of the following transitions represent?

a. $\delta(p, a, Z) = (q, aZ)$
b. $\delta(p, a, Z) = (q, \varepsilon)$
c. $\delta(p, a, Z) = (q, r)$
d. $\delta(p, \varepsilon, Z) = (q, r)$
e. $\delta(p, \varepsilon, \varepsilon) = (q, Z)$
f. $\delta(p, \varepsilon, Z) = (q, \varepsilon)$

The transition

$$\delta(p, a, Z) = (q, aZ)$$

means that the PDA in current state p after scanning the input symbol $a \in \Sigma$ and if $Z \in \Gamma$ is on top of the stack, then the PDA enters into new state q pushing $a \in \Sigma$ on to the stack. The transition

$$\delta(p, a, Z) = (q, \varepsilon)$$

means that in state p, on scanning the input symbol a, and when top of this stack is Z, the machine enters into state q and the topmost symbol Z is deleted from the stack. The transition

$$\delta(p, a, Z) = (q, r)$$

means that in state p, on scanning the input symbol *a* and when top of the stack is Z, the machine enters into state q and topmost symbol Z on the stack is replaced by r. The transition

$$\delta (p, \in, Z) = (q, r)$$

means that in state p, on scanning the empty string and when top of the stack is Z, the machine enters into q and topmost symbol Z on the stack is replaced by r. The transition

$$\delta(p, \varepsilon, \varepsilon) = (q, Z)$$

means that in state p, on scanning the empty string and when top of the stack empty, the machine enters into q pushing the symbol Z on the stack . The transition

$$\delta(p, \varepsilon, Z) = (q, \varepsilon)$$

means that in state p, on scanning the empty string and when top of the stack is Z, the machine enters into q and topmost symbol Z on the stack is deleted.

Note: δ is a transition function with three arguments. The first two are same as NFA i.e., the state and either ε or a symbol from the input alphabet. The third argument is the symbol on top of the stack. As the input symbol is *consumed* when the transition(or function) is applied, the symbol on top of the stack may be deleted or it may be altered or some times one or more items may be pushed on to the stack (depends on the problem being solved).

The **move** of a machine can be defined as follows.

Definition: Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. The move of machine M

$$\delta(p, a, Z) = (q, \alpha)$$

imply that, when the PDA is currently in state p, if the scanned input symbol is *a* and the top of the stack is Z, the PDA enters into new state q and pushes Z on the top of the stack. The symbol on top of the stack and the recently pushed symbol is denoted by α. Here $p, q \in Q$, $a \in \Sigma$, $Z \in \Gamma$ and $\alpha \in \Gamma^*$.

7.2 Graphical representation of a PDA

The various actions performed by a PDA in state *q* on an input symbol *a* and when the top of the stack represented by Z can be represented using two methods:

1. Using δ notation (as discussed in previous section)
2. Using graphical representation

Now, let us discuss how a PDA is represented graphically using the notations that have been used to represent an FA in which:

a. The states of the PDA correspond to the nodes in a graph and are represented using circles
b. The start state of the PDA is denoted by an arrow labeled *start*

c. The nodes of the graph represented by two concentric circles are the final states of the PDA
d. If there is a transition of the form

$$\delta(p, a, Z) = (q, \alpha)$$

then, there will be an arc from state p to state q and the arc is labeled with

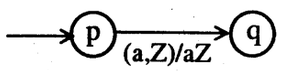
$$a, Z/\alpha$$

indicating a is the current symbol, Z is the symbol on top of the stack and α represent the top of the stack along with the recently pushed symbol.

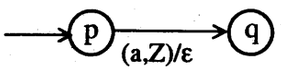
Example 7.3: For each of the transitions obtain the corresponding transition diagrams?

a. $\delta(p, a, Z) = (q, aZ)$
b. $\delta(p, a, Z) = (q, \varepsilon)$
c. $\delta(p, a, Z) = (q, r)$
d. $\delta(p, \varepsilon, Z) = (q, r)$
e. $\delta(p, \varepsilon, \varepsilon) = (q, Z)$
f. $\delta(p, \varepsilon, Z) = (q, \varepsilon)$

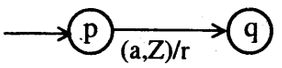
The transition $\delta(p, a, Z) = (q, aZ)$ means that the PDA in current state p after scanning the input symbol $a \in \Sigma$ and if $Z \in \Gamma$ is on top of the stack, then the PDA enters into new state q pushing $a \in \Sigma$ on to the stack. The graphical representation is shown below:



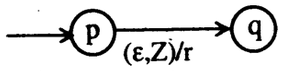
The transition $\delta(p, a, Z) = (q, \varepsilon)$ means that in state p, on scanning the input symbol a, and when top of this stack is Z, the machine enters into state q and the topmost symbol Z is deleted from the stack. The corresponding graphical representation is:



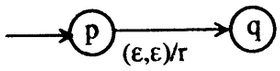
The transition $\delta(p, a, Z) = (q, r)$ means that in state p, on scanning the input symbol a and when top of the stack is Z, the machine enters into state q and topmost symbol Z on the stack is replaced by r. The corresponding graphical notation is shown below:



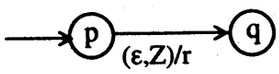
The transition $\delta(p, \varepsilon, Z) = (q, r)$ means that in state p, on scanning the empty string and when top of the stack is Z, the machine enters into q and topmost symbol Z on the stack is replaced by r and the equivalent graphical representation is shown below:



The transition $\delta(p, \varepsilon, \varepsilon) = (q, Z)$ means that in state p, on scanning the empty string and when top of the stack empty, the machine enters into q pushing the symbol Z on the stack. The equivalent graphical representation is shown below:



The transition $\delta(p, \varepsilon, Z) = (q, \varepsilon)$ means that in state p, on scanning the empty string and when top of the stack is Z, the machine enters into q and top most symbol Z on the stack is deleted. The graphical representation for this is shown below:



7.3 Instantaneous description

The current configuration of PDA at any given instant can be described by an *instantaneous description* (in short we can call ID). An ID gives the current state of the PDA, the remaining string to be processed and the entire contents of the stack. Thus, an instantaneous description ID can be defined as shown below.

Definition: Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. An ID (instaneous description) is defined as 3-tuple or a triple

$$(q, w, \alpha)$$

where q is the current state, *w* is the string to be processed and α is the current contents of stack. The leftmost symbol in the string α is on top of the stack and rightmost symbol in α is at the bottom of the stack.

Let the current configuration of PDA be

$$(q, aw, Z\alpha)$$

It means

- q - is the current state
- aw - is the string to be processed
- $Z\alpha$ - is the current contents of the stack with Z as the topmost symbol on the stack.

If the transition defined is $\delta(q, a, Z) = (p, \beta)$ then the new configuration obtained will be

$$(p, w, \beta\alpha)$$

The move from the current configuration to next configuration is given by

$$(q, aw, Z\alpha) \vdash (p, w, \beta\alpha)$$

This can be read as "the configuration $(q, aw, Z\alpha)$ derives $(p, w, \beta\alpha)$ in one move". If an arbitrary number of moves are used to move from one configuration to another configuration then the moves are denoted by the symbol $\vdash^*$ and $\vdash^+$.

For example,

$$(q, aw, Z\alpha) \vdash^* (p, w, \beta\alpha)$$

means that the current configuration of the PDA will be $(q, aw, Z\alpha)$ and after applying zero or more number of transitions, the PDA enters into new configuration $(p, w, \beta\alpha)$.

Note: The instantaneous description $(q, aw, Z\alpha) \vdash^+ (p, w, \beta\alpha)$ indicates that the configuration $(p, w, \beta\alpha)$ is obtained from the configuration $(q, aw, Z\alpha)$ by applying one or more transitions.

7.4 Acceptance of a language by PDA

There are two cases wherein a string *w* is accepted by a PDA.

- Get the final state from the start state.
- Get an empty stack from the start state

In the first case, we say that the language is accepted by a **final state** and in the second case we say that the language is accepted by an **empty stack** or **null stack**. The formal definitions to accept the string by a final state and by an empty stack are defined as follows.

Definition: Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. The language L(M) accepted by a final state is defined as

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

for some $\alpha \in \Gamma^*$, $p \in F$ and $w \in \Sigma^*$. It means that the PDA, currently in state q_0, after scanning the string *w* enters into a final state p. Once the machine is in state p, the input symbol should be ε and the contents of the stack are irrelevant. Any thing can be there on the stack. The only point to remember here is that when all the symbols in string *w* have been read and when the machine is in the final state the final contents of the stack are irrelevant.

We can also define a language N(M) accepted by an empty stack (Null stack) as

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

for $w \in \Sigma^*$, q_0, $p \in Q$. It means that when the string w is accepted by an empty stack, the final state is irrelevant, the input should be completely read and the stack should be empty. Here the state p is not the final state, only thing is that the string w should be completely read and stack should be empty.

7.5 Construction of PDA

Now, so far we have seen some concepts on PDA. In this section, we shall see how PDA's can be constructed.

Example 7.4: Obtain a PDA to accept the language $L(M) = \{wCw^R \mid w \in (a + b)^*\}$ where W^R is reveres of W by a final state.

It is clear from the language $L(M) = \{wCw^R\}$ that if

$$w = abb$$

then reverse of w denoted by w^R will be

$$w^R = bba$$

and the language L will be wCw^R i.e.,

$$abbCbba$$

which is a string of palindrome. So, we have to construct a PDA which accepts a palindrome consisting of *a*'s anb *b*'s with the symbol C in the middle.

General Procedure: To check for the palindrome, let us push all scanned symbols onto the stack till we encounter the letter C. Once we pass the middle string, if the string is a palindrome, for each scanned input symbol, there should be a corresponding symbol (same as input symbol) on the stack. Finally, if there is no input and stack is empty, we say that the given string is a palindrome.

Step1: Input symbols can be *a* or *b*.

Let q_0 be the initial state and Z_0 be the initial symbol on the stack. In state q_0 and when top of the stack is Z_0, whether the input symbol is *a* or *b* push it on to the stack, and remain in q_0. The transitions defined for this can be of the form

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$

Once the first scanned input symbol is pushed on to the stack, the stack may contain either *a* or *b*. Now, in state q_0, the input symbol can be either *a* or *b*. Note that irrespective of what is the input or what is there on the stack, we have to keep pushing all the symbols on to the stack, till we encounter C. So, the transitions defined for this can be of the form

$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, a) = (q_0, ba)$$
$$\delta(q_0, a, b) = (q_0, ab)$$
$$\delta(q_0, b, b) = (q_0, bb)$$

Step 2: Input symbol is C.

Now, if the next input symbol is C, the top of the stack may be *a* or *b*. Another possibly is that, in state q_0, the first symbol itself can be C. In this case *w* is null string and Z_0 will be on the stack. In all these cases, the input symbol is C i.e., the symbol which is present in the middle of the string. So, change the state to q_1 and do not alter the contents of the stack. The transitions defined for this can be of the form

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

Now, we have passed the middle of the string.

Step3: Input symbols can be *a* or *b*.

To be a palindrome, for each input symbol there should be a corresponding symbol (same as input symbol) on the stack. So, whenever the input symbol is same as symbol on the stack, remain in state q_1 and delete that symbol from the stack and repeat the process. The transitions defined for this can be of the form

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

Step 4: Finally, in state q_1, if the string is a palindrome, there is no input symbol to be scanned and the stack should be empty i.e., the stack should contain Z_0. Now, change the state to q_2 and do not alter the contents of the stack. The transition for this can be of the form

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

So, the PDA M to accept the language

$$L(M) = \{wCw^R \mid w \in (a,b)^*\}$$

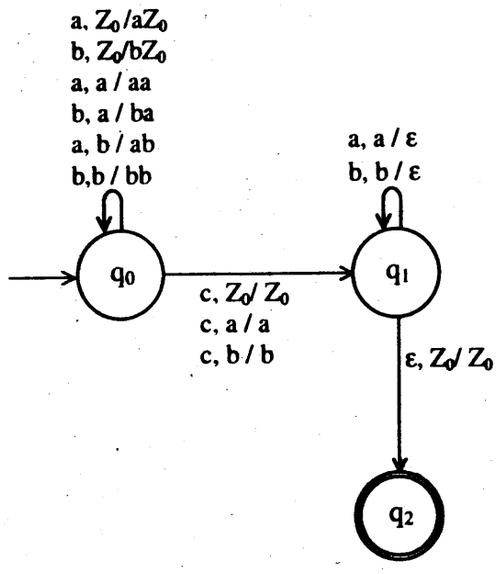
along with transition graph is given by

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

where

$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b, C\}$
$\Gamma = \{a, b, Z_0\}$
δ : is shown below.

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, b, Z_0) = (q_0, bZ_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_0, ba)$
$\delta(q_0, a, b) = (q_0, ab)$
$\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, c, Z_0) = (q_1, Z_0)$
$\delta(q_0, c, a) = (q_1, a)$
$\delta(q_0, c, b) = (q_1, b)$
$\delta(q_1, a, a) = (q_1, \varepsilon)$
$\delta(q_1, b, b) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$



$q_0 \in Q$ is the start state of machine.
$Z_0 \in \Gamma$ is the initial symbol on the stack.
$F = \{q_2\}$ is the final state.

To accept the string: The sequence of moves made by the PDA for the string aabCbaa is shown below.

Initial ID
$(q_0, aabCbaa, Z_0) \vdash (q_0, abCbaa, aZ_0)$
$\vdash (q_0, bCbaa, aaZ_0)$
$\vdash (q_0, Cbaa, baaZ_0)$
$\vdash (q_1, baa, baaZ_0)$
$\vdash (q_1, aa, aaZ_0)$
$\vdash (q_1, a, aZ_0)$
$\vdash (q_1, \varepsilon, Z_0)$
$\vdash (q_2, \varepsilon, Z_0)$
(Final Configuration)

Since q2 is the final state and input string is ε in the final configuration, the string

aabCbaa

is accepted by the PDA.

To reject the string: The sequence of moves made by the PDA for the string aabCbab is shown below.

Initial ID

$(q_0, aabCbab, Z_0) \vdash (q_0, abCbab, aZ_0)$

$\vdash (q_0, bCbab, aaZ_0)$

$\vdash (q_0, Cbab, baaZ_0)$

$\vdash (q_1, bab, baaZ_0)$

$\vdash (q_1, ab, aaZ_0)$

$\vdash (q_1, b, aZ_0)$

(Final Configuration)

Since the transition $\delta(q_1, b, a)$ is not defined, the string

aabCbab

is not a palindrome and the machine halts and the string is rejected by the PDA.

Note: The same problem can be converted to accept the language by an empty stack. Only the change is, instead of the final transition namely

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

replace it by the transition

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

When a language is accepted by an empty stack, finally the stack should not contain any thing including Z_0. Note that q_1 is not a final state. There is no final state.

Example 7.5: Obtain a PDA to accept the language $L = \{a^n b^n \mid n \geq 1\}$ by a final state.

Note: The machine should accept *n* number of *a*'s followed by *n* number of *b*'s.

General Procedure: Since *n* number of *a*'s should be followed by *n* number of *b*'s, let us push all the symbols on to the stack as long as the scanned input symbol is *a*. Once we encounter *b*'s, we should see that for each *b* in the input, there should be corresponding *a* on the stack. When the input pointer reaches the end of the string, the stack should be empty. If stack is empty, it indicates that the string scanned has *n* number of *a*'s followed by *n* number of *b*'s.

Step 1: Let q_0 be the start state and Z_0 be the initial symbol on the stack. As long as the next input symbol to be scanned is *a*, irrespective of what is there on the stack, keep pushing all the symbols on to the stack and remain in q_0. The transitions defined for this can be of the form

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

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

Step 2: In state q_0, if the next input symbol to be scanned is *b* and if the top of the stack is *a*, change the state to q_1 and delete one *b* from the stack. The transition for this can be of the form

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

Step 3: Once the machine is in state q_1, the rest of the symbols to be scanned will be only *b*'s and for each *b* there should be corresponding symbol *a* on the stack. So, as the scanned input symbol is *b* and if there is a matching *a* on the stack, remain in q_1 and delete the corresponding *a* from the stack. The transitions defined for this can be of the form

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

Step 4: In state q_1, if the next input symbol to be scanned is $\in$ and if the top of the stack is Z_0, (it means that for each *b* in the input there exists corresponding *a* on the stack) change the state to q_2 which is an accepting state. The transition defined for this can be of the form

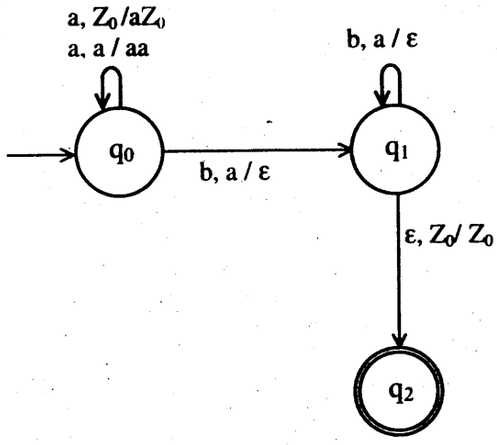
$$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$$

So, the PDA to accept the language

$$L = \{a^n b^n \mid n \geq 1\}$$

along with transition diagram is given by

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



where

$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, Z_0\}$
δ : is shown below.

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, a) = (q_1, \varepsilon)$$
$$\delta(q_1, b, a) = (q_1, \varepsilon)$$
$$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$$

$q_0 \in Q$ is the start state of machine..
$Z_0 \in \Gamma$ is the initial symbol on the stack.
$F = \{q_2\}$ is the final state.

To accept the string: The sequence of moves made by the PDA for the string aaabbb is shown below.

Initial ID
$(q_0, aaabbb, Z_0)$ $\vdash (q_0, aabbb, aZ_0)$
$\vdash (q_0, abbb, aaZ_0)$
$\vdash (q_0, bbb, aaaZ_0)$
$\vdash (q_1, bb, aaZ_0)$
$\vdash (q_1, b, aZ_0)$
$\vdash (q_1, \varepsilon, Z_0)$
$\vdash (q_2, \varepsilon, Z_0)$
(Final Configuration)

Since q2 is the final state and input string is ε in the final configuration, the string

aaabbb

is accepted by the PDA.

To reject the string: The sequence of moves made by the PDA for the string aabbb is shown below.

Initial ID
$(q_0, aabbb, Z_0)$ $\vdash (q_0, abbb, aZ_0)$
$\vdash (q_0, bbb, aaZ_0)$
$\vdash (q_1, bb, aZ_0)$
$\vdash (q_1, b, Z_0)$
(Final Configuration)

Since the transition $\delta(q_1, b, Z_0)$ is not defined, the string

aabbb

is rejected by the PDA.

Note: By changing the final transition from

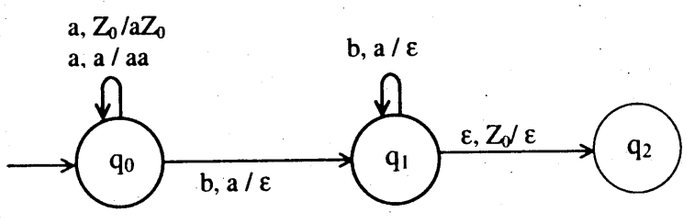
$$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$$

to

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

the PDA accepted by an empty stack is obtained. Note that q_2 is not the final state.

The corresponding transition diagram accepting by an empty stack is shown below:



Example 7.6: Obtain a PDA to accept the language L(M) = {w | w ∈ (a+b)* and $n_a(w) = n_b(w)$ by a final state i.e., number of a's in string w should be equal to number of b's in w.

The language accepted by the machine should consist strings of *a*'s and *b*'s of any length. Only restriction is that number of *a*'s in string *w* should be equal to number of *b*'s. The order of *a*'s and *b*'s is irrelevant. For example ε, ab, ba, aaabbb, ababab, aababb etc. are all the strings in the language L(M).

General Procedure: The first scanned input symbol is pushed on to the stack. From this point onwards, if the scanned symbol is same as the symbol on to the stack, push the current input symbol on to the stack. If the input symbol is different from the symbol on the top of the stack, pop one symbol from the stack and repeat the process. Finally, when end of string is encountered, if the stack is empty, the string *w* has equal number of *a*'s and *b*'s, otherwise number of *a*'s and *b*'s are different.

Step 1: Let q_0 be the start state and Z_0 be the initial symbol on the stack. When the machine is in state q_0 and when top of the stack contains Z_0, scan the input symbol (either *a* or *b*) and push it on to the stack. The transitions defined for this can be of the form

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$

Step 2: Once the first input symbol is pushed on to the stack, the top of stack may contain either *a* or *b* and the next input symbol to be scanned may be *a* or *b*. If the input symbol is same as the symbol on top of the stack, push the current input symbol on to the stack and remain in state q_0 only. Otherwise, pop an element from the stack. The transitions defined for this can be of the form

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

Step 3: In state q_0, if the next symbol to be scanned is ε (empty string) and top of the stack is Z_0, it means that for each symbol *a* there exists a corresponding *b* and for each symbol *b*, there exists

a symbol *a*. So, the string *w* consists of equal number of *a*'s and *b*'s and change the state to q_1. The transition defined for this can be of the form

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

So, the PDA to accept the language

$$L = \{w \mid n_a(w) = n_b(w)\}$$

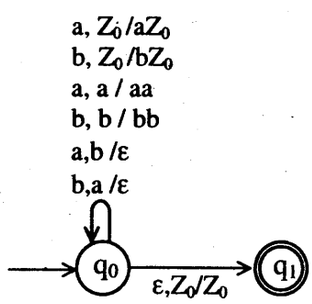
is given by

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

where

$Q = \{q_0, q_1\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, b, Z_0\}$
δ : is shown below.

$$\begin{aligned}
\delta(q_0, a, Z_0) &= (q_0, aZ_0) \\
\delta(q_0, b, Z_0) &= (q_0, bZ_0) \\
\delta(q_0, a, a) &= (q_0, aa) \\
\delta(q_0, b, b) &= (q_0, bb) \\
\delta(q_0, a, b) &= (q_0, \varepsilon) \\
\delta(q_0, b, a) &= (q_0, \varepsilon) \\
\delta(q_0, \varepsilon, Z_0) &= (q_1, Z_0)
\end{aligned}$$



$q_0 \in Q$ is the start state of machine.
$Z_0 \in \Gamma$ is the initial symbol on the stack.
$F = \{q_1\}$ is the final state.

To accept the string: The sequence of moves made by the PDA for the string abbbaa is shown below.

Initial ID
$(q_0, abbbaa, Z_0)$ $\vdash (q_0, bbbaa, aZ_0)$
$\vdash (q_0, bbaa, Z_0)$
$\vdash (q_0, baa, bZ_0)$
$\vdash (q_0, aa, bbZ_0)$
$\vdash (q_0, a, bZ_0)$
$\vdash (q_0, \varepsilon, Z_0)$
$\vdash (q_1, \varepsilon, Z_0)$
(Final Configuration)

Since q_1 is the final state and input string is ε in the final configuration, the string

abbbaa

is accepted by the PDA.

To reject the string: The sequence of moves made by the PDA for the string aabbb is shown below.

Initial ID

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

$\vdash$ (q_0, bbb, aaZ_0)

$\vdash$ (q_0, bb, aZ_0)

$\vdash$ (q_0, b, Z_0)

$\vdash$ (q_0, ε, bZ_0)

(Final Configuration)

Since the transition $\delta(q_0, \varepsilon, b)$ is not defined, the string

aabbb

is rejected by the PDA.

Note: To accept the language by an empty stack, the final state is irrelevant where as the next input symbol to be scanned should be ε and stack should to be empty. Even Z_0 should not be there on the stack. So, to obtain the PDA to accept equal number of *a*'s and *b*'s using an empty stack, change only the last transition in example 7.5. The last transition

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

can be changed as

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

So, the PDA to accept the language

$$L = \{w \mid n_a(w) = n_b(w)\}$$

by an empty stack is given by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varepsilon)$$

where

$Q = \{q_0, q_1\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, b, Z_0\}$
δ : is shown below.

$$\begin{aligned}
\delta(q_0, a, Z_0) &= (q_0, aZ_0) \\
\delta(q_0, b, Z_0) &= (q_0, bZ_0) \\
\delta(q_0, a, a) &= (q_0, aa) \\
\delta(q_0, b, b) &= (q_0, bb) \\
\delta(q_0, a, b) &= (q_0, \varepsilon) \\
\delta(q_0, b, a) &= (q_0, \varepsilon) \\
\delta(q_0, \varepsilon, Z_0) &= (q_1, \varepsilon)
\end{aligned}$$

$q_0 \in Q$ is the start state of machine.
$Z_0 \in \Gamma$ is the initial symbol on the stack.
$F = \{\phi\}$. Note that PDA is accepted by an empty stack

The sequence of moves made by the PDA for the string aabbab by an empty stack is shown below.

Initial ID
$(q_0, aabbab, Z_0)$ $\vdash (q_0, abbab, aZ_0)$
$\vdash (q_0, bbab, aaZ_0)$
$\vdash (q_0, bab, aZ_0)$
$\vdash (q_0, ab, Z_0)$
$\vdash (q_0, b, aZ_0)$
$\vdash (q_0, \varepsilon, Z_0)$
$\vdash (q_1, \varepsilon, \varepsilon)$
(Final Configuration)

Since the next input symbol to be scanned is ε and the stack is empty, the string aabbab is accepted by an empty stack.

Note: A PDA accepting by a final state and by an empty stack are equivalent. They can be obtained from each other i.e., if we know a PDA by a final state, we can obtain a PDA by an empty stack and vice versa.

Example 7.7: Obtain a PDA to accept a string of balanced parentheses. The parentheses to be considered are (,), [,]

Note 1: Some of the valid strings are:

[( ) ( ) ([ ])], ε, [ ] [ ] [( )

and invalid strings are:

[ ) ( ) [ ],) (, [ )

Note 2: ε (null string) is valid

Note 3: Left parentheses can either be '(' or '[' and right parentheses can either be ')' or ']'.

Step 1: Let q_0 be the start state and Z_0 be the initial symbol on the stack. The state q_0 itself is the final state accepting ε (an empty string).

Step 2: In state q_0, if the first scanned parentheses is '(' or '[', push the scanned symbol on to the stack and change the state to q_1. The transition defined for this can be of the form

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

Step 3: If at least one parentheses either '(' or '[' is present on the stack and if the scanned symbol is left parentheses, then push the left parentheses on to the stack. The transitions defined for this can be of the form

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

Step 4: If the scanned symbol is ')' and if the top of the stack is '(' pop an element from the stack. Similarly, if the scanned symbol is ']' and if the top of the stack is '[' pop an element from the stack. The transitions defined for this can be of the form

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

Step 5: When top of the stack is Z_0, it indicates that so far all the parentheses have been matched. At this point, on ε-transition, the PDA enters into state q0 and all the steps from step1 are repeated. The transition for this can be of the form

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

So, the PDA to accept the language consisting of balanced parentheses is given by

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

where

$Q = \{q_0, q_1\}$
$\Sigma = \{(,), [,]\}$
$\Gamma = \{(, [, Z_0\}$
δ : is shown below.

$$\begin{aligned}
\delta(q_0, (, Z_0) &= (q_1, (Z_0) \\
\delta(q_0, [, Z_0) &= (q_1, [Z_0) \\
\delta(q_1, (, () &= (q_1, (() \\
\delta(q_1, (, [) &= (q_1, ([) \\
\delta(q_1, [, () &= (q_1, [() \\
\delta(q_1, [, [) &= (q_1, [[) \\
\delta(q_1,), (&= (q_1, \varepsilon) \\
\delta(q_1,], [&= (q_1, \varepsilon) \\
\delta(q_1, \varepsilon, Z_0) &= (q_0, Z_0)
\end{aligned}$$

$q_0 \in Q$ is the start state of machine.
$Z_0 \in \Gamma$ is the initial symbol on the stack.
$F = \{q_0\}$. Note that even ε is accepted by PDA and is valid.

The sequence of moves made by the PDA for the string [( )( )([ ])] is shown below.

Initial ID

$$\begin{aligned}
(q_0, [(\,)(\,)(\,[\,]\,)\,], Z_0) &\vdash (q_1, (\,)(\,)(\,[\,]\,)\,], [Z_0) \\
&\vdash (q_1, \;\;)(\,)(\,[\,]\,)\,], ([\,Z_0) \\
&\vdash (q_1, \;\;\;(\,)(\,[\,]\,)\,], [\,Z_0) \\
&\vdash (q_1, \;\;\;\;)(\,[\,]\,)\,], ([\,Z_0) \\
&\vdash (q_1, \;\;\;\;\;(\,[\,]\,)\,], [\,Z_0) \\
&\vdash (q_1, \;\;\;\;\;\;[\,]\,)\,], ([\,Z_0) \\
&\vdash (q_1, \;\;\;\;\;\;\;]\,)\,], [\,([\,Z_0) \\
&\vdash (q_1, \;\;\;\;\;\;\;\;)\,], ([\,Z_0) \\
&\vdash (q_1, \;\;\;\;\;\;\;\;\;], [\,Z_0) \\
&\vdash (q_1, \varepsilon, Z_0) \\
&\vdash (q_0, \varepsilon, Z_0) \\
&\quad \text{(Final Configuration)}
\end{aligned}$$

Since the next input symbol to be scanned is ε and the stack is empty, the string [( )( )([ ])] is accepted by the PDA.

Example 7.8: Obtain a PDA to accept the language $L = \{w \mid w \in (a, b)^* \text{ and } n_a(w) > n_b(w)\}$

Note: The solution is similar to that of the problem discussed in example 7.5 in which we are accepting strings of *a*'s and *b*'s of equal numbers. When we encounter end of the input i.e., ε and top of the stack is Z_0, it has equal number of *a*'s and *b*,s. But, what we want is a machine to accept more number of *a*'s than *b*'s. For this, only change to be made is that when we encounter ε (i.e., end of the input), if top of the stack contains at least one *a*, then change the final state to q1 and do not alter the contents of the stack. The transition defined remains same as problem shown in example 7.5, except the last transition. The last transition is of the form

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

So, the PDA to accept the language

$$L = \{w \mid n_a(w) > n_b(w)\}$$

is given by

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

where

$Q = \{q_0, q_1\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, b, Z_0\}$
δ : is shown below.

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$
$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, b) = (q_0, bb)$$
$$\delta(q_0, a, b) = (q_0, \varepsilon)$$
$$\delta(q_0, b, a) = (q_0, \varepsilon)$$
$$\delta(q_0, \varepsilon, a) = (q_1, a)$$

$q_0 \in Q$ is the start state of machine.
$Z_0 \in \Gamma$ is the initial symbol on the stack.
$F = \{q_1\}$ is the final state.

Note: On similar lines we can find a PDA to accept the language

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

i.e., strings of *a*'s and *b*'s where number of *b*'s are more than number of *a*'s. To achieve this only change to be made in the above machine is change the final transition

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

to

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

So, the PDA to accept the language

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

is given by

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

where

$Q = \{q_0, q_1\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, b, Z_0\}$
δ : is shown below.

$$\begin{aligned}
\delta(q_0, a, Z_0) &= (q_0, aZ_0) \\
\delta(q_0, b, Z_0) &= (q_0, bZ_0) \\
\delta(q_0, a, a) &= (q_0, aa) \\
\delta(q_0, b, b) &= (q_0, bb) \\
\delta(q_0, a, b) &= (q_0, \varepsilon) \\
\delta(q_0, b, a) &= (q_0, \varepsilon) \\
\delta(q_0, \varepsilon, b) &= (q_1, b)
\end{aligned}$$

$q_0 \in Q$ is the start state of machine.
$Z_0 \in \Gamma$ is the initial symbol on the stack.
$F = \{q_1\}$ is the final state.

Example 7.9: Obtain a PDA to accept the language $L = \{a^n b^{2n} \mid n \geq 1\}$

Note: The machine should accept *n* number of *a*'s followed by 2*n* number of *b*'s.

General Procedure

Since *n* number of *a*'s should be followed by 2*n* number of *b*'s, for each *a* in the input, push two *a*'s on to the stack. Once we encounter *b*'s, we should see that for each *b* in the input, there should be corresponding *a* on the stack. When the input pointer reaches the end of the string, the stack should be empty. If stack is empty, it indicates that the string scanned has *n* number of *a*'s followed by 2*n* number of *b*'s.

Step 1: Let q_0 be the start state and Z_0 be the initial symbol on the stack. For each scanned input symbol *a*, push two *a*'s on to the stack. The transitions defined for this can be of the form

$$\begin{aligned}
\delta(q_0, a, Z_0) &= (q_0, aaZ_0) \\
\delta(q_0, a, a) &= (q_0, aaa)
\end{aligned}$$

Step 2: In state q_0, if the next input symbol to be scanned is *b* and if the top of the stack is *a*, change the state to q_1 and delete one *b* from the stack. The transition for this can be of the form

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

Step 3: Once the machine is in state q_1, the rest of the symbols to be scanned will be only *b*'s and for each *b* there should be corresponding symbol *a* on the stack. So, as the scanned input symbol is *b* and if there is a matching *a* on the stack, remain in q_1 and delete the corresponding *a* from the stack. The transitions defined for this can be of the form

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

Step 4: In state q_1, if the next input symbol to be scanned is $\in$ and if the top of the stack is Z_0, (it means that for each *b* in the input there exists corresponding *a* on the stack) change the state to q_2 which is an accepting state. The transition defined for this can be of the form

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

So, the PDA to accept the language

$$L = \{a^n b^{2n} \mid n \geq 1\}$$

is given by

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

where

$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, Z_0\}$
δ : is shown below.

$$\delta(q_0, a, Z_0) = (q_0, aaZ_0)$$
$$\delta(q_0, a, a) = (q_0, aaa)$$
$$\delta(q_0, b, a) = (q_1, \varepsilon)$$
$$\delta(q_1, b, a) = (q_1, \varepsilon)$$
$$\delta(q_1, \varepsilon, Z_0) = (q_2, \varepsilon)$$

$q_0 \in Q$ is the start state of machine.
$Z_0 \in \Gamma$ is the initial symbol on the stack.
$F = \{q_2\}$ is the final state.

To accept the string: The sequence of moves made by the PDA for the string aabbbb is shown below.

Initial ID
$(q_0, aabbbb, Z_0)$ $\vdash$ $(q_0, abbbb, aaZ_0)$
$\vdash$ $(q_0, bbbb, aaaaZ_0)$
$\vdash$ $(q_0, bbb, aaaZ_0)$
$\vdash$ (q_1, bb, aaZ_0)
$\vdash$ (q_1, b, aZ_0)
$\vdash$ (q_1, ε, Z_0)
$\vdash$ (q_2, ε, Z_0)
(Final Configuration)

Since q_2 is the final state and input string is ε in the final configuration, the string

aabbbb

is accepted by the PDA.

To reject the string: The sequence of moves made by the PDA for the string aabbb is shown below.

Initial ID
$(q_0, aabbb, Z_0)$ $\vdash$ $(q_0, abbb, aaZ_0)$
$\vdash$ $(q_0, bbb, aaaaZ_0)$
$\vdash$ $(q_0, bb, aaaZ_0)$
$\vdash$ (q_0, b, aaZ_0)
$\vdash$ (q_0, ε, aZ_0)
(Final Configuration)

Since the transition $\delta(q_0, \varepsilon, a)$ is not defined, the string

aabbb

is rejected by the PDA.

Example 7.10: Obtain a PDA to accept the language $L = \{ww^R \mid w \in (a + b)^*\}$ by a final state

It is clear from the language $L(M) = \{ww^R\}$ that if

$w = abb$

then reverse of w denoted by w^R will be

$w^R = bba$

and the language L will be ww^R i.e.,

abbbba

which is a string of palindrome.

So, we have to construct a PDA which accepts a palindrome consisting of *a*'s anb *b*'s. This problem is similar to the problem discussed in example 7.3. Only difference is that in example 7.3, an extra symbol C acts as a pointer to the middle string. But, here there is no way to find the mid point for the string.

General Procedure: To check for the palindrome, let us push all scanned symbols onto the stack till we encounter the mid point (Remember that there is no way to find the midpoint). Once we pass the middle string, to be a palindrome, for each scanned input symbol, there should be a corresponding symbol(same as input symbol) on the stack. Finally, if there is no input and stack is empty, we say that the given string is a palindrome.

Step1: Let q_0 be the initial state and Z_0 be the initial symbol on the stack. In state q_0 and when top of the stack is Z_0, whether the input symbol is *a* or *b* push it on to the stack, and remain in q_0. The transitions defined for this can be of the form

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$

Note: In state q_0, if we encounter ε, the empty string has to be accepted and we should be in a position to reach the final state and so, we may have one more transition of the form

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

Once the first scanned input symbol is pushed on to the stack, the stack may contain either *a* or *b*. Now, in state q_0, the input symbol can be either *a* or *b*. Note that irrespective of what is the input or what is there on the stack, we have to keep pushing all the symbols on to the stack, till we encounter midpoint (But, there is no way to find mid point. We continue this process till we encounter mid point through our common sense).

So, the transitions defined for this can be of the form

$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, b, a) = (q_0, ba)$$
$$\delta(q_0, a, b) = (q_0, ab)$$
$$\delta(q_0, b, b) = (q_0, bb)$$

Step 2: Now, once we reach the midpoint, the top of the stack may be *a* or *b*. To be a palindrome, for each input symbol there should be a corresponding symbol (same as input symbol) on the stack. So, whenever the input symbol is same as symbol on the stack, change the state to q_1 and delete that symbol from the stack The transitions defined for this can be of the form

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

Step 3: Now, once we are in state q_1, it means that we have passed the mid point. Now, the top of the stack may be *a* or *b*. To be a palindrome, for each input symbol there should be a corresponding symbol (same as input symbol) on the stack. So, whenever the input symbol is same as symbol on the stack, remain in state q_1 and delete that symbol from the stack. The transitions defined for this can be of the form

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

Step 4: Finally, in state q_1, if the string is a palindrome, there is no input symbol to be scanned and the stack should be empty i.e., the stack should contain Z_0. Now, change the state to q_2 and do not alter the contents of the stack. The transition for this can be of the form

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

So, the PDA M to accept the language

$$L(M) = \{ww^R \mid w \in (a.b)^*\}$$

is given by

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

where

$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, b, Z_0\}$
δ : is shown below.

	$\delta(q_0, \varepsilon, Z_0)$	=	(q_1, Z_0)
	$\delta(q_0, a, Z_0)$	=	(q_0, aZ_0)
	$\delta(q_0, b, Z_0)$	=	(q_0, bZ_0)
3	$\delta(q_0, a, a)$	=	(q_0, aa)
	$\delta(q_0, b, a)$	=	(q_0, ba)
	$\delta(q_0, a, b)$	=	(q_0, ab)
6	$\delta(q_0, b, b)$	=	(q_0, bb)
7	$\delta(q_0, a, a)$	=	(q_1, ε)
8	$\delta(q_0, b, b)$	=	(q_1, ε)
	$\delta(q_1, a, a)$	=	(q_1, ε)
	$\delta(q_1, b, b)$	=	(q_1, ε)
	$\delta(q_1, \varepsilon, Z_0)$	=	(q_2, Z_0)

$q_0 \in Q$ is the start state of machine.
$Z_0 \in \Gamma$ is the initial symbol on the stack.
$F = \{q2\}$ is the final state.

Note that the transitions numbered 3 and 7, 6 and 8 can be combined and the transitions can be written as shown below also.

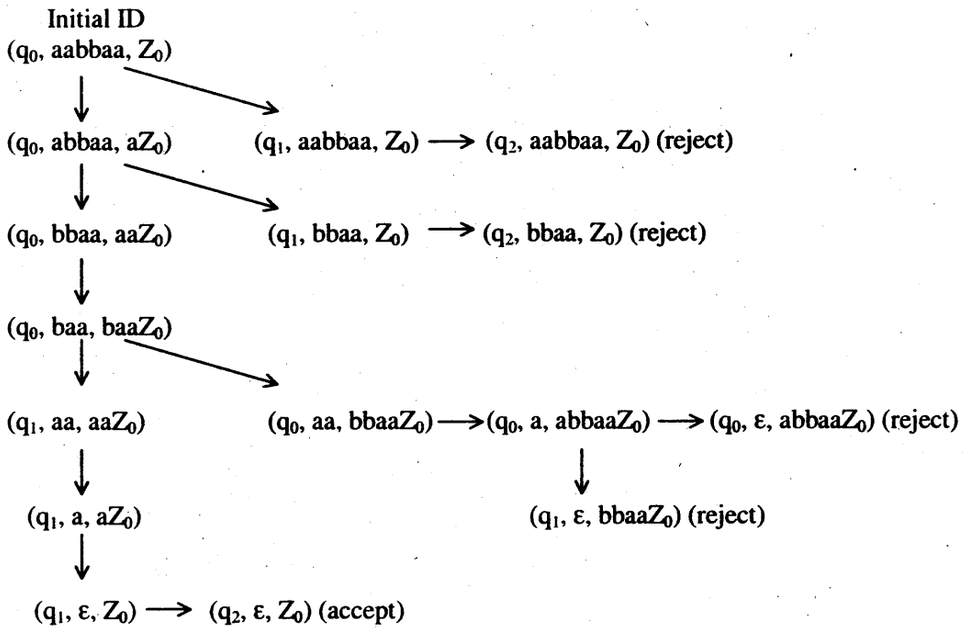
$\delta(q_0, \varepsilon, Z_0)$	=	(q_1, Z_0)
$\delta(q_0, a, Z_0)$	=	(q_0, aZ_0)
$\delta(q_0, b, Z_0)$	=	(q_0, bZ_0)
$\delta(q_0, a, a)$	=	$\{ (q_0, aa), (q_1, \varepsilon) \}$
$\delta(q_0, b, a)$	=	(q_0, ba)
$\delta(q_0, a, b)$	=	(q_0, ab)
$\delta(q_0, b, b)$	=	$\{ (q_0, bb), (q_1, \varepsilon) \}$
$\delta(q_1, a, a)$	=	(q_1, ε)
$\delta(q_1, b, b)$	=	(q_1, ε)
$\delta(q_1, \varepsilon, Z_0)$	=	(q_2, Z_0)

Note that once the following transitions are applied

$\delta(q_0, a, a) = \{ (q_0, aa), (q_1, \varepsilon) \}$
$\delta(q_0, b, b) = \{ (q_0, bb), (q_1, \varepsilon) \}$

if the input symbol is same as the symbol on top of the stack, the machine may push the current symbol on to the stack, or it may pop an element from the stack. At this point, the machine makes appropriate decision so that if the string is a palindrome, it has to accept. This machine is clearly a non-deterministic PDA (in short we call NPDA).

To accept the string: The sequence of moves made by the PDA for the string aabbaa is shown below.



Since q_2 is the final state and input string is ε in the final configuration, the string *aabbaa* is accepted by the PDA.

7.6 Deterministic and Non-deterministic PDA

In the example 7.10, we have seen that there can be multiple transitions defined from a state. The PDA defined in example 7.10, is clearly a non-deterministic PDA. Now, let us see the difference between *deterministic* PDA and *non-deterministic* PDA.

Definition: Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. The PDA is deterministic if

1. $\delta(q, a, Z)$ has only one element.
2. If $\delta(q, \varepsilon, Z)$ is not empty, then $\delta(q, a, Z)$ should be empty.

Both the conditions should be satisfied for the PDA to be deterministic. If one of the conditions fails, the PDA is non-deterministic.

In the PDAs discussed in the examples 7.3 to 7.10, let us see what are deterministic PDAs and what are non-deterministic PDAs.

Example 7.11: Is the PDA to accept the language L(M) = {wCw^R | w∈ (a + b)*} discussed in example 7.4 is deterministic?

The transitions defined for this machine are obtained in example 7.4 and are reproduced for the sake of convenience.

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, b, Z_0) = (q_0, bZ_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_0, ba)$
$\delta(q_0, a, b) = (q_0, ab)$
$\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, c, Z_0) = (q_1, Z_0)$
$\delta(q_0, c, a) = (q_1, a)$
$\delta(q_0, c, b) = (q_1, b)$
$\delta(q_1, a, a) = (q_1, \varepsilon)$
$\delta(q_1, b, b) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$

The PDA should satisfy the two conditions shown in the definition to be deterministic.

1. $\delta(q, a, Z)$ has only one element: Note that in the transitions, for each $q \in Q$, $a \in \Sigma$ and $Z \in \Gamma$, there is only one component defined and the first condition is satisfied.
2. The second condition states that if $\delta(q, \varepsilon, Z)$ is not empty, then $\delta(q, a, Z)$ should be empty i.e., If there is an ε-transition, (in this case it is $\delta(q_1, \varepsilon, Z_0)$), then there should not be any transition from the state q_1 when top of the stack is Z_0 which is true.

Since, the PDA satisfies both the conditions, the PDA is deterministic.

Example 7.12: Is the PDA corresponding to the language L = {$a^n b^n$ | n≥1} by a final state is deterministic?

The transitions defined for this machine discussed in the example 7.5 are reproduced here for the sake of convenience.

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \varepsilon)$
$\delta(q_1, b, a) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, Z_0) = (q_2, \varepsilon)$

The first condition to be deterministic is $\delta(q, a, Z)$ should have only one component. In this case, for each $q \in Q$, $a \in \Sigma$ and $Z \in \Gamma$, there exists only one definition. So, the first condition is satisfied. To satisfy the second condition, consider the transition

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

Since the transition is defined, the transition $\delta(q_1, a, Z_0)$ where $a \in \Sigma$ should not be defined which is true. Since both the conditions are satisfied, the given PDA is deterministic.

Example 7.13: Is the PDA to accept the language $L(M) = \{w \mid w \in (a+b)^*$ and $n_a(w) = n_b(w)$ is deterministic?

The transitions defined for this machine discussed in the example 7.6 are reproduced here for the sake of convenience.

$$\begin{aligned}
\delta(q_0, a, Z_0) &= (q_0, aZ_0)\\
\delta(q_0, b, Z_0) &= (q_0, bZ_0)\\
\delta(q_0, a, a) &= (q_0, aa)\\
\delta(q_0, b, b) &= (q_0, bb)\\
\delta(q_0, a, b) &= (q_0, \varepsilon)\\
\delta(q_0, b, a) &= (q_0, \varepsilon)\\
\delta(q_0, \varepsilon, Z_0) &= (q_1, Z_0)
\end{aligned}$$

The first condition to be deterministic is $\delta(q, a, Z)$ should have only one component. In this case, for each $q \in Q$, $a \in \Sigma$ and $Z \in \Gamma$, there exists only one component. So, the first condition is satisfied. To satisfy the second condition, consider the transition

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

Since this transition is defined, the transition $\delta(q_0, a, Z_0)$ where $a \in \Sigma$ should not be defined. But, there are two transitions

$$\begin{aligned}
\delta(q_0, a, Z_0) &= (q_0, aZ_0)\\
\delta(q_0, b, Z_0) &= (q_0, bZ_0)
\end{aligned}$$

defined from q_0 when top of the stack is Z_0. Since the second condition is not satisfied, the given PDA is *non-deterministic* PDA.

Example 7.14: Is the PDA to accept the language consisting of balanced parentheses is deterministic?

The transitions defined for this machine discussed in the example 7.7 are reproduced here for the sake of convenience.

$$\begin{aligned}
\delta(q_0, (, Z_0) &= (q_1, (Z_0)\\
\delta(q_0, [, Z_0) &= (q_1, [Z_0)\\
\delta(q_1, (, ( ) &= (q_1, (( )
\end{aligned}$$

$$\begin{aligned}
\delta(q_1, (, [\,) &= (q_1, ([\,)\\
\delta(q_1, [, (\,) &= (q_1, [(\,)\\
\delta(q_1, [, [\,) &= (q_1, [[\,)\\
\delta(q_1,), (\,) &= (q_1, \varepsilon)\\
\delta(q_1,], [\,) &= (q_1, \varepsilon)\\
\delta(q_1, \varepsilon, Z_0) &= (q_0, Z_0)
\end{aligned}$$

The first condition to be deterministic is $\delta(q, a, Z)$ should have only one component. In this case, for each $q \in Q$, $a \in \Sigma$ and $Z \in \Gamma$, there exists only one component. So, the first condition is satisfied. To satisfy the second condition, consider the transition

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

Since this transition is defined, the transition $\delta(q_1, a, Z_0)$ where $a \in \Sigma$ should not be defined which is true. Since both the conditions are satisfied, the given PDA is deterministic.

Example 7.15: Is the PDA to accept the language $L = \{w \mid w \in (a, b)^*$ and $n_a(w) > n_b(w)\}$ is deterministic?

The transitions defined for this machine discussed in the example 7.8 are reproduced here for the sake of convenience.

$$\begin{aligned}
\delta(q_0, a, Z_0) &= (q_0, aZ_0)\\
\delta(q_0, b, Z_0) &= (q_0, bZ_0)\\
\delta(q_0, a, a) &= (q_0, aa)\\
\delta(q_0, b, b) &= (q_0, bb)\\
\delta(q_0, a, b) &= (q_0, \varepsilon)\\
\delta(q_0, b, a) &= (q_0, \varepsilon)\\
\delta(q_0, \varepsilon, a) &= (q_1, a)
\end{aligned}$$

The first condition to be deterministic is $\delta(q, a, Z)$ should have only one component. In this case, for each $q \in Q$, $a \in \Sigma$ and $Z \in \Gamma$, there exists only one component. So, the first condition is satisfied. To satisfy the second condition, consider the transition

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

Since this transition is defined, the transition $\delta(q_0, f, a)$ where $f \in \Sigma$ should not be defined. But, there are two transitions

$$\begin{aligned}
\delta(q_0, a, a) &= (q_0, aa)\\
\delta(q_0, b, a) &= (q_0, \varepsilon)
\end{aligned}$$

defined from q_0 when top of the stack is *a*. Since the second condition is not satisfied, the given PDA is *non-deterministic* PDA.

Example 7.16: Is the PDA to accept the language $L = \{a^n b^{2n} \mid n \geq 1\}$ is deterministic?

The transitions defined for this machine discussed in the example 7.9 are reproduced here for the sake of convenience.

$$\begin{aligned}
\delta(q_0, a, Z_0) &= (q_0, aaZ_0) \\
\delta(q_0, a, a) &= (q_0, aaa) \\
\delta(q_0, b, a) &= (q_1, \varepsilon) \\
\delta(q_1, b, a) &= (q_1, \varepsilon) \\
\delta(q_1, \varepsilon, Z_0) &= (q_2, \varepsilon)
\end{aligned}$$

The first condition to be deterministic is $\delta(q, a, Z)$ should have only one component. In this case, for each $q \in Q$, $a \in \Sigma$ and $Z \in \Gamma$, there exists only one definition. So, the first condition is satisfied. To satisfy the second condition, consider the transition

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

Since the transition is defined, the transition $\delta(q_1, a, Z_0)$ where $a \in \Sigma$ should not be defined which is true. Since both the conditions are satisfied, the given PDA is deterministic.

Example 7.17: Is the PDA to accept the language $L = \{ww^R \mid w \in (a + b)^*\}$ is deterministic?

The transitions defined for this machine discussed in the example 7.10 are reproduced here for the sake of convenience.

$$\begin{aligned}
\delta(q_0, a, Z_0) &= (q_0, aZ_0) \\
\delta(q_0, b, Z_0) &= (q_0, bZ_0) \\
\delta(q_0, a, a) &= (q_0, aa), (q_1, \varepsilon) \\
\delta(q_0, b, a) &= (q_0, ba) \\
\delta(q_0, a, b) &= (q_0, ab) \\
\delta(q_0, b, b) &= (q_0, bb), (q_1, \varepsilon) \\
\delta(q_1, a, a) &= (q_1, \varepsilon) \\
\delta(q_1, b, b) &= (q_1, \varepsilon) \\
\delta(q_1, \varepsilon, Z_0) &= (q_2, Z_0)
\end{aligned}$$

The first condition to be deterministic is $\delta(q, a, Z)$ should have only one component. But, there are two transitions each having two components viz.,

$$\begin{aligned}
\delta(q_0, a, a) &= (q_0, aa), (q_1, \varepsilon) \\
\delta(q_0, b, b) &= (q_0, bb), (q_1, \varepsilon)
\end{aligned}$$

So, the first condition fails. To be deterministic, both the conditions shown in the definition should be satisfied. Since one of the conditions is not met, the PDA is *non-deterministic*.

7.7 CFG to PDA

It is quite easy to get a PDA from the context free grammar. This is possible only from a CFG which is in GNF. So, given any grammar, first obtain the grammar in GNF and then obtain the PDA. The steps to be followed to convert a grammar to its equivalent PDA are shown below.

1. Convert the grammar into GNF
2. Let q_0 be the start state and Z_0 is the initial symbol on the stack. Without consuming any input, push the start symbol S onto the stack and change the state to q_1. The transition for this can be
$$\delta(q_0, \varepsilon, Z_0) = (q_1, SZ_0)$$
3. For each production of the form
$$A \rightarrow a\alpha$$
introduce the transition
$$\delta(q_1, a, A) = (q_1, \alpha)$$
4. Finally, in state q_1, without consuming any input, change the state to q_f which is an accepting state. The transition for this can be of the form
$$\delta(q_1, \varepsilon, Z_0) = (q_f, Z_0)$$

Example 7.18: For the grammar

$S \rightarrow aABC$
$A \rightarrow aB|a$
$B \rightarrow bA|b$
$C \rightarrow a$

Obtain the corresponding PDA

Let q_0 be the start state and Z_0 the initial symbol on the stack.
Step 1: Push the start symbol S on to the stack and change the state to q_1. The transition for this can be of the form
$$\delta(q_0, \varepsilon, Z_0) = (q_1, SZ_0)$$
Step 2: For each production $A \rightarrow a\alpha$ introduce the transition
$$\delta(q_1, a, A) = (q_1, \alpha)$$
This can be done as shown below.

Production	Transition
$S \rightarrow aABC$	$\delta(q_1, a, S) = (q_1, ABC)$
$A \rightarrow aB$	$\delta(q_1, a, A) = (q_1, B)$
$A \rightarrow a$	$\delta(q_1, a, A) = (q_1, \varepsilon)$
$B \rightarrow bA$	$\delta(q_1, b, B) = (q_1, A)$
$B \rightarrow b$	$\delta(q_1, b, B) = (q_1, \varepsilon)$
$C \rightarrow a$	$\delta(q_1, a, C) = (q_1, \varepsilon)$

Step 3: Finally in state q_1, without consuming any input change the state to q_f which is an accepting state i.e.,

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

So, the PDA M is given by $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where

$Q = \{q_0, q_1, q_f\}$
$\Sigma = \{a, b\}$
$\Gamma = \{S, A, B, C, Z_0\}$
δ : is shown below.

$\delta(q_0, \varepsilon, Z_0) = (q_1, SZ_0)$
$\delta(q_1, a, S) = (q_1, ABC)$
$\delta(q_1, a, A) = (q_1, B)$
$\delta(q_1, a, A) = (q_1, \varepsilon)$
$\delta(q_1, b, B) = (q_1, A)$
$\delta(q_1, b, B) = (q_1, \varepsilon)$
$\delta(q_1, a, C) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, Z_0) = (q_f, Z_0)$

$q_0 \in Q$ is the start state of machine.
$Z_0 \in \Gamma$ is the initial symbol on the stack.
$F = \{q_f\}$ is the final state

Note that the terminals of grammar G will be the input symbols in PDA and the non-terminals will be the stack symbols in PDA.The derivation from the grammar is shown below:

$S \Rightarrow aABC$
$\Rightarrow aaBBC$
$\Rightarrow aabBC$
$\Rightarrow aabbC$
$\Rightarrow aabba$

The string aabba is derived from the start symbol S. The same string should be accepted by PDA also. The moves made by the PDA are shown below.

Initial ID		
$(q_0, aabba, Z_0)$	$\vdash (q_1, aabba, SZ_0)$	By Rule 1
	$\vdash (q_1, abba, ABCZ_0)$	By Rule 2
	$\vdash (q_1, bba,, BBCZ_0)$	By Rule 3
	$\vdash (q_1, ba, BCZ_0)$	By Rule 6
	$\vdash (q_1, a, CZ_0)$	By Rule 6
	$\vdash (q_1, \varepsilon, Z_0)$	By Rule 7
	$\vdash (q_f, \varepsilon, Z_0)$	By Rule 8 (Final configuration)

Since q_f is the final state and input string is ε in the final configuration, the string

aabba

is accepted by the PDA.

Example 7.19: For the grammar

$$\begin{aligned} S &\rightarrow aABB \mid aAA \\ A &\rightarrow aBB \mid a \\ B &\rightarrow bBB \mid A \\ C &\rightarrow a \end{aligned}$$

Obtain the corresponding PDA

Note: To obtain a PDA from CFG, the grammar should be in GNF. All the productions except the production

$$B \rightarrow A$$

are in GNF. Since one of the production is not in GNF, the given grammar is not in GNF. This can be easily converted into GNF. Note that all A productions are in GNF. So, by substituting for A in the above production we get B-productions also in GNF as shown below:

$$B \rightarrow bBB \mid aBB \mid a$$

Now, the new grammar which is in GNF can take the form:

$$\begin{aligned} S &\rightarrow aABB \mid aAA \\ A &\rightarrow aBB \mid a \\ B &\rightarrow bBB \mid aBB \mid a \\ C &\rightarrow a \end{aligned}$$

Now, the CFG can be converted into PDA. Let q_0 be the start state and Z_0 the initial symbol on the stack.

Step 1: Push the start symbol S on to the stack and change the state to q_1. The transition for this can be of the form

$$\delta(q_0, \varepsilon, Z_0) = (q_1, SZ_0)$$

Step 2: For each production $A \rightarrow a\alpha$ introduce the transition

$$\delta(q_1, a, A) = (q_1, \alpha)$$

The following table shows how the various transtitions are obtained from the productions from the CFG.

Production	Transition
$S \rightarrow aABB$	$\delta(q_1, a, S) = (q_1, ABB)$
$S \rightarrow aAA$	$\delta(q_1, a, S) = (q_1, AA)$
$A \rightarrow aBB$	$\delta(q_1, a, A) = (q_1, BB)$
$A \rightarrow a$	$\delta(q_1, a, A) = (q_1, \varepsilon)$
$B \rightarrow bBB$	$\delta(q_1, b, B) = (q_1, BB)$
$B \rightarrow aBB$	$\delta(q_1, a, B) = (q_1, BB)$
$B \rightarrow a$	$\delta(q_1, a, B) = (q_1, \varepsilon)$
$C \rightarrow a$	$\delta(q_1, a, C) = (q_1, \varepsilon)$

Step 3: Finally in state q_1, without consuming any input change the state to q_f which is an accepting state i.e.,

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

So, the PDA M is given by $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
where

$Q = \{q_0, q_1, q_f\}$
$\Sigma = \{a, b\}$
$\Gamma = \{S, A, B, C, Z_0\}$
δ : is shown below.

$$\begin{aligned}
\delta(q_0, \varepsilon, Z_0) &= (q_1, SZ_0)\\
\delta(q_1, a, S) &= (q_1, ABB)\\
\delta(q_1, a, S) &= (q_1, AA)\\
\delta(q_1, a, A) &= (q_1, BB)\\
\delta(q_1, a, A) &= (q_1, \varepsilon)\\
\delta(q_1, b, B) &= (q_1, BB)\\
\delta(q_1, a, B) &= (q_1, BB)\\
\delta(q_1, a, B) &= (q_1, \varepsilon)\\
\delta(q_1, a, C) &= (q_1, \varepsilon)\\
\delta(q_1, \varepsilon, Z_0) &= (q_f, Z_0)
\end{aligned}$$

$q_0 \in Q$ is the start state of machine.
$Z_0 \in \Gamma$ is the initial symbol on the stack.
$F = \{q_f\}$ is the final state

Note that the terminals of grammar G will be the input symbols in PDA and the non-terminals will be the stack symbols in PDA. It is left to the reader to derive the string from grammar G and to show that the same string can be generated by the PDA which can be done similar to the previous problem.

7.8 Application of GNF

Now, we can easily say where the GNF notations are used. In the previous sections we have seen how easily a PDA can be obtained if the grammar is in GNF compared to the section 7.5. In section 7.5, we have designed various types of PDAs but with little bit of difficulty. The same PDAs could have been designed by obtaining the CFG and then convert the CFG into GNF. Once the grammar is in GNF, we can easily obtain PDA.

Note: Instead of using the method shown in section 7.5, it is much easier to obtain CFG, convert it into GNF and then obtain the PDA. In chapter 1 we have obtained the CFGs to accept varieties of languages. Obtain the GNF notation for those grammars and use the approach given in previous section, to obtain the PDAs.

This section provides some of the grammars and their equivalent PDA just to show how easy to obtain a PDA if the grammar is in GNF.

Example 7.20: Obtain a the PDA to accept the language $L = \{a^n b^n \mid n \geq 1\}$

The equivalent CFG to generate the language is:

$$S \rightarrow aSb$$
$$S \rightarrow ab$$

The corresponding grammar which is in GNF is

$$S \rightarrow aSB$$
$$S \rightarrow aB$$
$$B \rightarrow b$$

The equivalent transitions for the above productions are:

$$\delta(q_1, a, S) = (q_1, SB)$$
$$\delta(q_1, a, S) = (q_1, B)$$
$$\delta(q_1, b, B) = (q_1, \varepsilon)$$

In section 7.7, we have seen that the transition

$$\delta(q_0, \varepsilon, Z_0) = (q_1, SZ_0)$$

is used to push S on to the stack initially and the last transition

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

is used to move to the final state. So, the PDA is given by

$$M = (\{q_0, q_1, q_f\}, \{a,b\}, \{S,A,B,Z_0\}, \delta, q_0, Z_0, q_f)$$