UNIT 3

Syllabus Context-Free Grammars: Definition of Context-Free Grammars, Derivations Using a Grammar, Leftmost and Rightmost Derivations, the Language of a Grammar, Sentential Forms, Parse Tress, Applications of Context-Free Grammars, Ambiguity in Grammars and Languages. Push Down Automata,: Definition of the Pushdown Automaton, the Languages of a PDA, Equivalence of PDA's and CFG's, Deterministic Pushdown Automata.

==========

Context-Free Grammar

Definition – A context-free grammar (CFG) consisting of a finite set of grammar productions

Grammar G is represented as (N, T, P, S) where

- N is a set of non-terminal symbols.
- T is a set of terminals where N ∩ T = NULL.
- P is a set of rules, P: N → (N ∪ T)*,
- i.e., the left-hand side of the production contains only one nonterminal and right hand side of the production contains any set of terminals and non-terminals.
- S is the start symbol.

Example

- The grammar ({A}, {a, b, c}, P, A), P : A → aA, A → abc.
- The grammar ({S, a, b}, {a, b}, P, S), P: S → aSa, S → bSb, S → ε
- The grammar ({S, F}, {0, 1}, P, S), P: S → 00S | 11F, F → 00F | ε

Derivation:

Definition: The process of generating a string from starting symbol of the grammar is called as "Derivation".

Derivation always starts from Starting symbol of the grammar

Then each nonterminal is replaced with its right side of the productions.

Example:

Let a CFG {N,T,P,S},

N = {S}, T = {a, b}, Starting symbol = S, P = S → SS | aSb | ε

One derivation from the above CFG is "abaabb"

S → SS → aSbS → abS → abaSb → abaaSbb → abaabb

Leftmost and Rightmost Derivations

Basically there are two types of derivations in context free grammar. They are

- **Leftmost derivation** – A leftmost derivation is obtained by applying production to the leftmost variable in each step.
- **Rightmost derivation** – A rightmost derivation is obtained by applying production to the rightmost variable in each step.

Example

Let any set of production rules in a CFG be

X → X+X | X*X |X| a over an alphabet {a}.

The **leftmost derivation** for the string **a+a*a** will be –

X → X+X → a+X → a + X*X → a+a*X → a+a*a

The **rightmost derivation** for the string **a+a*a** from above grammar is –

X → X*X → X*a → X+X*a → X+a*a → a+a*a

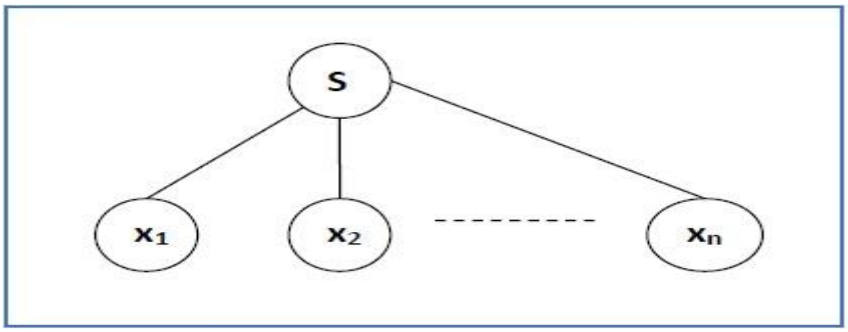
Derivation Tree

A **derivation tree or parse tree** is an ordered rooted tree that graphically represents the semantic information of a string derived from a context-free grammar.

Representation Technique

- Root vertex – Must be labeled by the start symbol.
- Vertex – Labeled by a non-terminal symbol.
- Leaves – Labeled by a terminal symbol or ε.

If $S \rightarrow x_1x_2 \ldots\ldots x_n$ is a production rule in a CFG, then the parse tree / derivation tree will be as follows –



Example

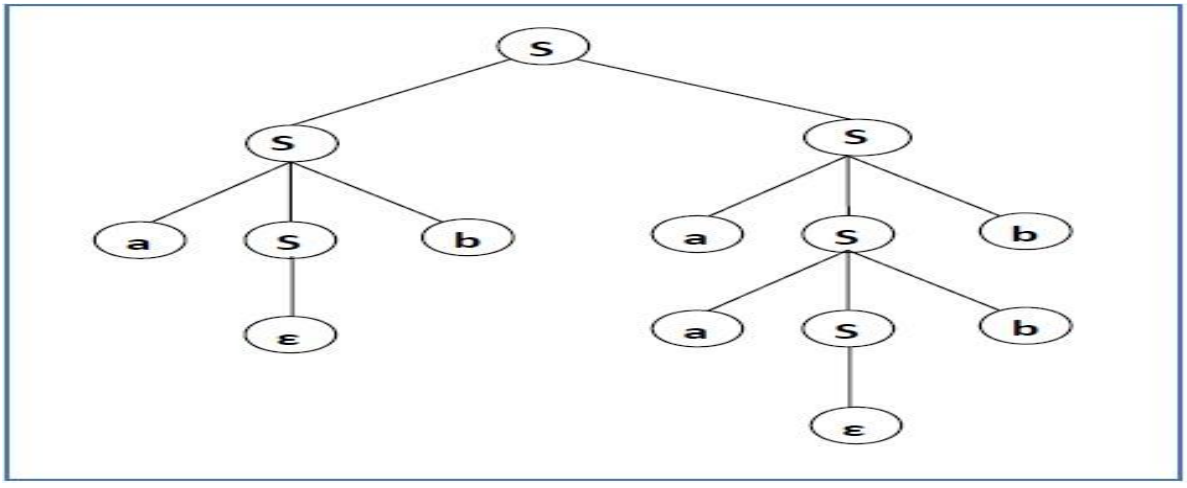
Let a CFG {N,T,P,S} be

N = {S}, T = {a, b}, Starting symbol = S, P = S → SS | aSb | ε

One derivation from the above CFG is "abaabb"

S → SS → aSbS → abS → abaSb → abaaSbb → abaabb

Derivation Tree for string **abaabb** is :



Ambiguity in Context-Free Grammars

If a context free grammar G has more than one derivation trees for some string w ∈ L(G), it is called an ambiguous grammar. There exist multiple right-most or left-most derivations for some string generated from that grammar.

Problem

Check whether the grammar G with production rules –

X → X+X | X*X |X| a

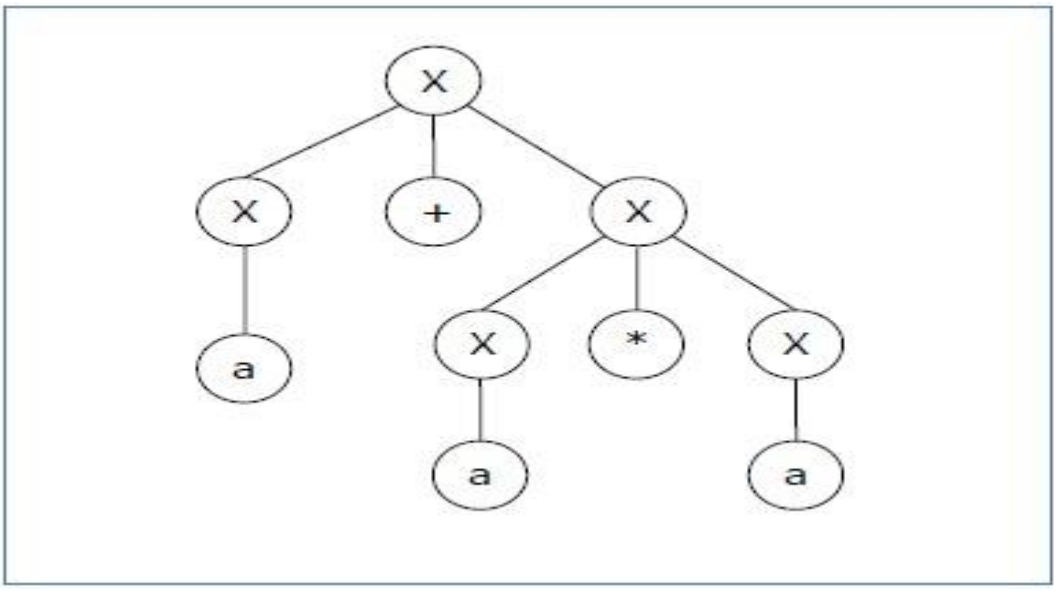
is ambiguous or not.

Solution

Let's find out the derivation tree for the string "a+a*a". It has two leftmost derivations.

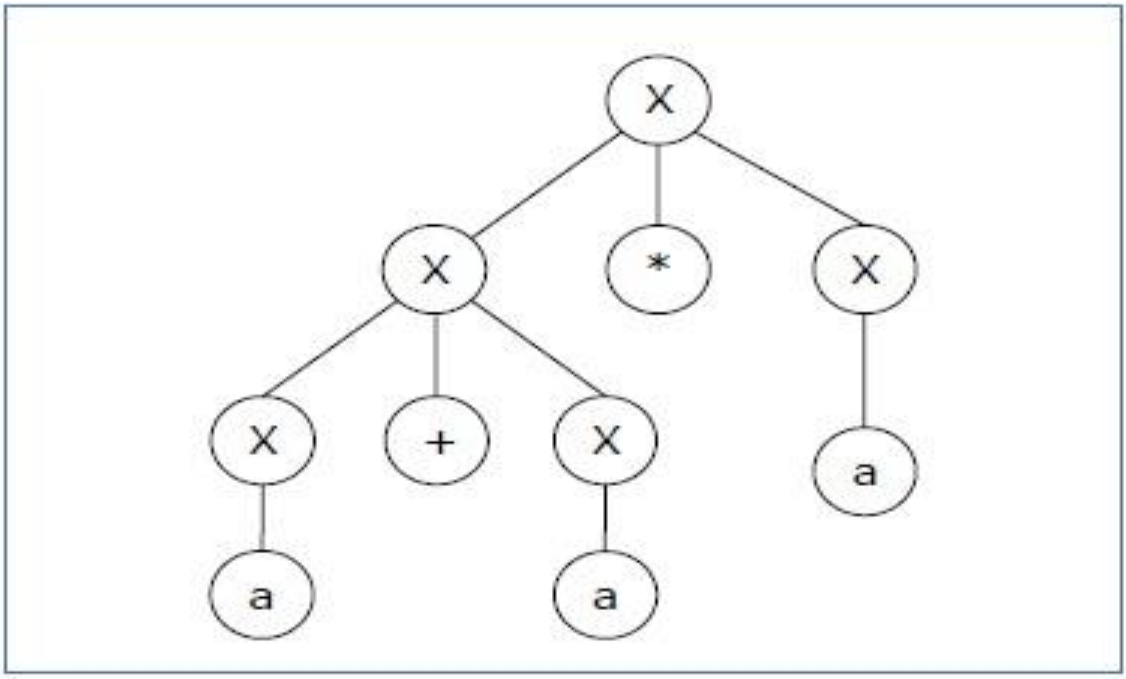
Derivation 1 – X → X+X → a +X → a+ X*X → a+a*X → a+a*a

Derivation tree or Parse tree 1



Derivation 2 – X → X*X → X+X*X → a+ X*X → a+a*X → a+a*a

Parse tree 2



Since there are two parse trees for a single string "a+a*a", then grammar **G is ambiguous.**

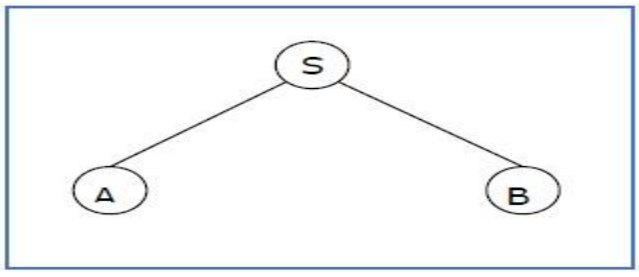
Sentential Form and Partial Derivation Tree

A partial derivation tree is a sub-tree of a derivation tree/parse tree such that either all of its children are in the sub-tree or none of them are in the sub-tree.

Example: If in any CFG the productions are –

S → AB, A → aaA | ε, B → Bb| ε

the partial derivation tree can be the following –



If a partial derivation tree contains the root S, it is called a **sentential form**.

Applications of Context Free Grammars:

Context Free Grammar (CFG) is of great practical importance. It is used for following purposes-

- For defining programming languages
- For translation of programming languages
- For describing arithmetic expressions
- For construction of compilers
- Data Processing
- Natural Language Processing
- Neural Networks
- Multi –Functional Radar Systems
- Human activities Recognition

Pushdown Automata Introduction

Basic Structure of PDA

A pushdown automaton is a way to implement a context-free grammar in a similar way we design DFA for a regular grammar. A DFA never remembers the information, but a PDA can remember an infinite amount of information.

Basically a pushdown automaton is –

"Finite state machine" + "a stack"

A pushdown automaton has three components –

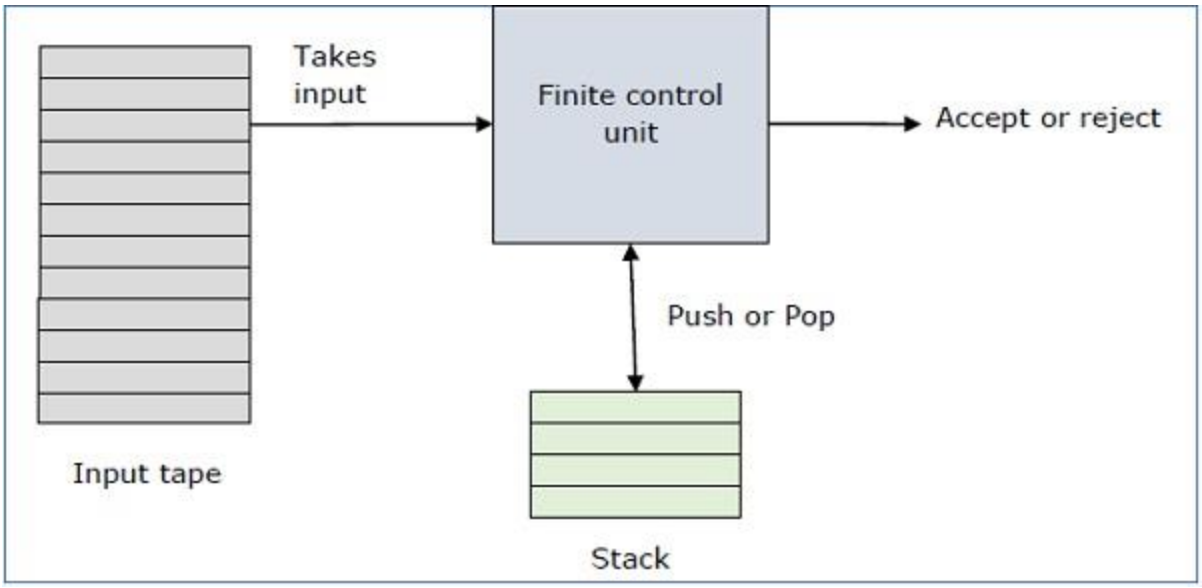
- an input tape,
- a control unit, and
- a stack with infinite size.

The stack head scans the top symbol of the stack.

A stack does three operations –

- **Push** – a new symbol is added at the top of stack.
- **Pop (ε)** – the top of stack symbol is read and removed.
- **No-change** – top of the stack symbol is read and that remains same.

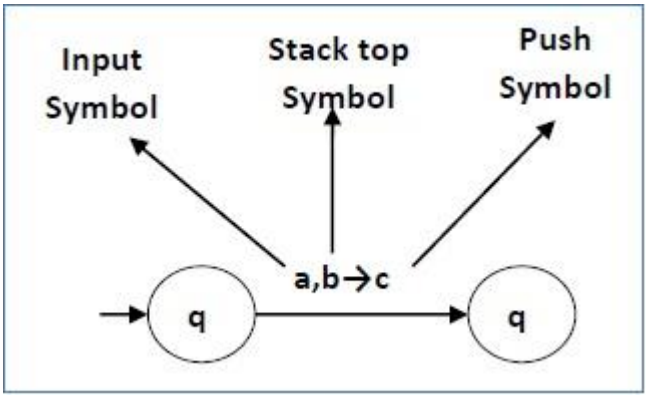
A PDA may or may not read an input symbol, but it has to read the top of the stack in every transition.



A PDA can be formally described as a 7-tuple (Q, Σ, S, δ, q_0, Z_0, F) –

- **Q** is the finite number of states
- **Σ** is input alphabet
- **S** is stack symbols
- **δ** is the transition function: Q × (Σ ∪ {ε}) × S → Q × S*
- **q_0** is the initial state (q0 ∈ Q)
- **Z_0** is the initial stack top symbol (Z_0 ∈ S)
- **F** is a set of accepting states (F ∈ Q)

The following diagram shows a transition in a PDA from a state q_1 to state q_2, labeled as a,b → c –



This means at state q_1, if we encounter an input string 'a' and top symbol of the stack is 'b', then we push 'c' on top of the stack 'b' and move to state q_2.

Terminologies Related to PDA

Instantaneous Description

The instantaneous description (ID) of a PDA is represented by a triplet (q, w, s) where

- **q** is the state
- **w** is unconsumed input
- **s** is the stack contents

Turnstile Notation

The "turnstile" notation is used for connecting pairs of ID's that represent one or many moves of a PDA. The process of transition is denoted by the turnstile symbol "⊢".

Consider a PDA (Q, Σ, S, δ, q0, Z_0, F). A transition can be mathematically represented by the following turnstile notation –

```
(p, aw, Tβ) ⊢ (q, w, αb)
```

This implies that while taking a transition from state p to state q, the input symbol 'a' is consumed, and the top of the stack 'T' is replaced by a new string 'α'.

Note – If we want zero or more moves of a PDA, we have to use the symbol (⊢*) for it.

Languages of PDA: A Pushdown automaton accepts context free language and regular language.

Now let's see how to construct PDA for languages.

1. Construct PDA on language that accepts L = {0^n 1^n | n ≥ 0} ?
Sol:

Here language generates strings as L = {ab, aabb, aaabbb, aaaabbbb, ...}

In each of the string, the number of a's are followed by equal number of b's.

Here, we need to maintain the order of a's and b's. That is, all the a's are coming first and then all the b's are coming. Thus, we need a stack along with the state diagram. The count of a's and b's is maintained by the stack. We will take 2 stack alphabets: S= { a, z }
Where, S= set of all the stack alphabet
z = stack start symbol

As we want to design a PDA, thus every time 'a' comes before 'b'.

When 'a' comes then push it in stack and if again 'a' comes then also push it. After that, when 'b' comes then pop one 'a' from the stack

each time. So, at the end if the stack becomes empty then we can say that the string is accepted by the PDA.

Stack transition functions –

$\delta(q0, a, z) \vdash (q0, az)$

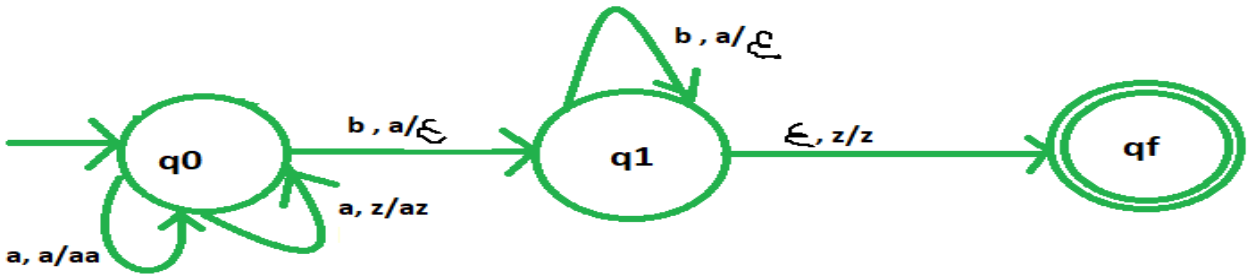
$\delta(q0, a, a) \vdash (q0, aa)$

$\delta(q0, b, a) \vdash (q1, \in)$

$\delta(q1, b, a) \vdash (q1, \in)$

$\delta(q1, \in, z) \vdash (qf, z)$

Where, q0 = Initial state, qf = Final state, $\in$=indicates pop operation



Required PDA

String acceptancy: Let check string **aabb** is accepted or not.

$\delta(q0,aabb, z) = (q0, az)$

$\delta(q0,abb, az) = (q0, aaz)$

$\delta(q0,bb, aaz) = (q1, az)$

$\delta(q1,b, az) = (q1, z)$

$\delta(q1, \in, z) = (qf)$

Finally stack empty and PDA reached to final State so we can conclude string **aabb** is accepted.

2. Construct PDA on language that accepts $L = \{a^n b^m c^n \mid m,n>=1\}$?

Sol:

Here, we need to maintain the order of a's, b's and c's.That is, all the a's are are coming first and then all the b's and then c's are coming. Thus, we need a stack along with the state diagram. The count of a's and c's is maintained by the stack. The number of a's is exactly equal to the number of c's.

Every time 'a' comes before 'b'. When 'a' comes then push it in stack and if again 'a' comes then also push it. And, when 'c' comes then pop one 'a' from the stack each time . And for 'b', we will do nothing in the stack only change the state in state diagram. So, at the end if the stack becomes empty then we can say that the string is accepted by the PDA.

Stack transition functions –

$\delta(q0, a, z) \vdash (q0, az)$

$\delta(q0, a, a) \vdash (q0, aa)$

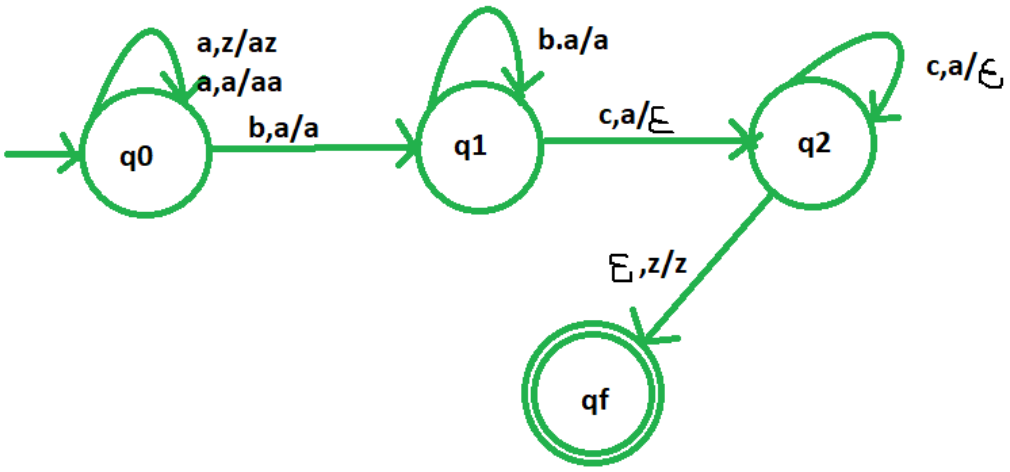
$\delta(q0, b, a) \vdash (q1, a)$

$\delta(q1, b, a) \vdash (q1, a)$

$\delta(q1, c, a) \vdash (q2, \in)$

$\delta(q2, c, a) \vdash (q2, \in)$

$\delta(q2, \in , z) \vdash (qf, z)$



Required PDA

Pushdown Automata Acceptance

There are two different ways to define PDA acceptability.

Final State Acceptability

In final state acceptability, a PDA accepts a string when, after reading the entire string, the PDA is in a final state. From the starting state, we can make moves that end up in a final state with any stack values. The stack values are irrelevant as long as we end up in a final state.

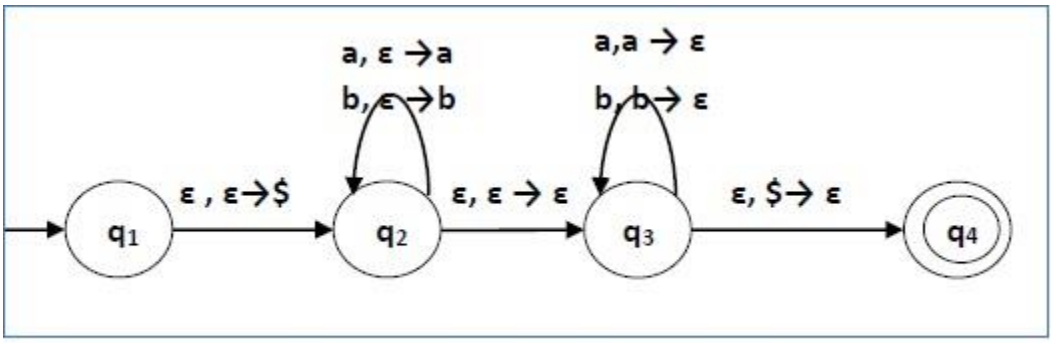
For a PDA (Q, Σ, S, δ, q0, Z_0, F), the language accepted by the set of final states F is –

L(PDA) = {w | (q0, w, I) ⊢* (q, ε, x), q ∈ F}

for any input stack string **x**.

Example: Construct a PDA that accepts L = { ww^R | w = (a+b)* }

Solution



PDA for L= {ww^R | w = (a+b)*}

Initially we put a special symbol '$' into the empty stack. At state **q_2**, the **w** is being read. In state **q_3**, each 0 or 1 is popped when it matches the input. If any other input is given, the PDA will go to a dead state. When we reach that special symbol '$', we go to the accepting state **q_4**.

Empty Stack Acceptability

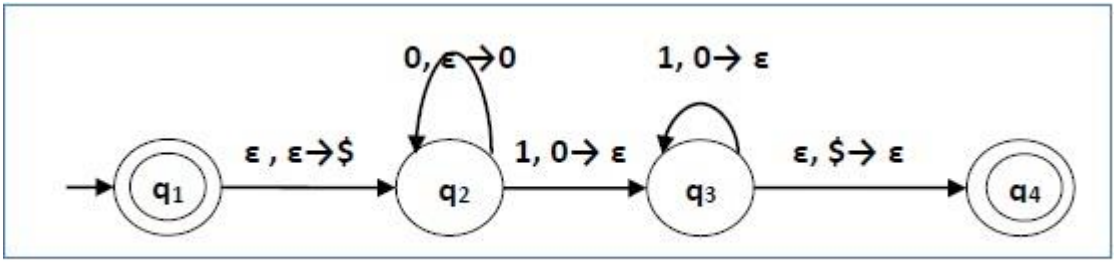
Here a PDA accepts a string when, after reading the entire string, the PDA has emptied its stack.

For a PDA (Q, Σ, S, δ, q0, I, F), the language accepted by the empty stack is –

L(PDA) = {w | (q0, w, I) ⊢* (q, ε, ε), q ∈ Q}

Example: Construct a PDA that accepts L = {$0^n 1^n$ | n ≥ 0}

Solution



PDA for L= {$0^n 1^n$ | n≥0}

This language accepts L = {ε, 01, 0011, 000111, ... }

Here, in this example, the number of **'a'** and **'b'** have to be same.

- Initially we put a special symbol **'$'** into the empty stack.
- Then at state **q_2**, if we encounter input 0 and top is Null, we push 0 into stack. This may iterate. And if we encounter input 1 and top is 0, we pop this 0.
- Then at state **q_3**, if we encounter input 1 and top is 0, we pop this 0. This may also iterate. And if we encounter input 1 and top is 0, we pop the top element.
- If the special symbol '$' is encountered at top of the stack, it is popped out and it finally goes to the accepting state q_4.

Equivalence of PDA and CFGs:

If a grammar G is context-free, we can build an equivalent nondeterministic PDA which accepts the language that is produced by the context-free grammar G.

Also, if P is a pushdown automaton, an equivalent context-free grammar G can be constructed where

L(G) = L(P)

In the next two topics, we will discuss how to convert from **PDA to CFG and CFG to PDA.**

From CFG to PDA:

Algorithm to find PDA corresponding to a given CFG

Input − A CFG, G = (V, T, P, S)

Output − Equivalent PDA, P = (Q, Σ, S, δ, q0, I, F)

Step 1 − Identify Non terminals & terminals in the CFG

Step 2 − The PDA will have only one state {q}.

Step 3 − The start symbol of CFG will be the start symbol in the PDA.

Step 4 − All non-terminals of the CFG will be the stack symbols of the PDA and all the terminals of the CFG will be the input symbols of the PDA.

Step 5 − For each Non terminal production in the grammar of the form $A \rightarrow aX$, where a is terminal and A, X are combination of terminal and non-terminals, make a transition

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

Step 6 − For each terminal in the grammar add new PDA transitions as

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

Problem

Construct a PDA from the following CFG.

G = ({S, X}, {a, b}, P, S) where the productions are −

S → XS | ε , A → aXb | Ab | ab

Solution

Let the equivalent PDA,

P = ({q}, {a, b}, {a, b, X, S}, δ, q, S)

where δ −

δ(q, ε , S) = {(q, XS), (q, ε)}

δ(q, ε , A) = {(q, aXb), (q, Ab), (q, ab)}

δ(q, a, a) = {(q, ε)}

δ(q, b, b) = {(q, ε)}

From PDA to CFG:

Algorithm to find CFG corresponding to a given PDA

Input − A CFG, G = (V, T, P, S)

Output − Equivalent PDA, P = (Q, Σ, S, δ, q0, Z_0, F)

The productions for CFG are written as follows:

Step 1: Starting Symbol of the grammar S production to be written as

S → [q0, Z_0 , q"] where

q0 is initial state of PDA,

Z_0 is initial Stack Symbol and

q" belongs to all states of given PDA

Step 2: For every action of Popping move in PDA, production to be written as

$$\delta (q ,a ,Z) = (q' , \varepsilon)$$ is a pop transition

where q, q' are either same or different states, a is current input symbol, Z is present top of the stack then pop operation is represented with ε.

Then its equivalent CFG production is

$$[q, Z, q'] \rightarrow a$$

Step 3: For every action of No change move in PDA, production to be written as

$$\delta (q ,a ,Z) = (q1 , Z)$$ is a no change transition

where q,q1 are different states, a is current input symbol, Z is present top of the stack then no change operation is represented with same current top of stack symbol.

Then its equivalent CFG production is

$$[q, Z, q''] \rightarrow a\ [q1, Z, q'']$$

Where q" is set of states in PDA.

Step 4: For every action of Insertion move in PDA, production to be written as

$$\delta (q ,a ,Z0) = (q1 , Z1Z0)$$ is a insertion transition

where q,q1 are different states, a is current input symbol, Z0 is present top of the stack, Z1 is new top of stack after the transition.

Then its equivalent CFG production is

$$[q, Z0, q''] \rightarrow a\ [q1, Z1, q']\ [q', Z0, q'']$$

Where q', q" are set of states in PDA.

Problem

Convert following PDA into CFG?

M = {{q0,q1}, {0,1}, {x,z0}, δ , q0, z0, q1} where δ is given as follows:

δ (q0 ,1 ,Z0) = (q0 , xZ0)

δ (q0 ,1 ,x) = (q0 , xx)

δ (q0 ,0 ,x) = (q0 , x)

δ (q0 , 1 ,x) = (q1 , ε)

δ (q1 , ε ,x) = (q1 , ε)

Sol:

Starting symbol productions for CFG according to Step 1:

S → [q0, Z_0 , q0] / [q0, Z_0 , q1]

Then consider first transition of PDA : δ (q0 ,1 ,Z0) = (q0 , xZ0)

It is an insertion operation so use step 4 , then its equialent productions are

[q0, Z0, q0] → 1 [q0, x, q0] [q0, Z0, q0]

[q0, Z0, q0] → 1 [q0, x, q1] [q1, Z0, q0]

[q0, Z0, q1] → 1 [q0, x, q0] [q0, Z0, q1]

[q0, Z0, q1] → 1 [q0, x, q1] [q1, Z0, q1]

δ (q0 ,1 ,x) = (q0 , xx)

[q0, x, q0] → 1 [q0, x, q0] [q0, x, q0]

[q0, x, q0] → 1 [q0, x, q1] [q1, x, q0]

[q0, x, q1] → 1 [q0, x, q0] [q0, x, q1]

[q0, x, q1] → 1 [q0, x, q1] [q1, x, q1]

δ (q0 ,0 ,x) = (q0 , x) : it is no change operation, so use step 3.

[q0, x, q0] → 0 [q0, x, q0]

[q0, x, q1] → 0 [q0, x, q1]

δ (q0 , 1 ,x) = (q1 , ε) : it is pop operation, so use step 2

[q0, x, q1] → 1

δ (q1 , ε ,x) = (q1 , ε)

[q1, x, q1] → ε