Chapter – 3

Pushdown Automata:

- Pushdown automata is a way to implement a CFG in the same way we design DFA for a regular grammar. A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information.
- Pushdown automata is simply an NFA augmented with an "external stack memory". The addition of stack is used to provide a last-in-first-out memory management capability to Pushdown automata. Pushdown automata can store an unbounded amount of information on the stack. It can access a limited amount of information on the stack. A PDA can push an element onto the top of the stack and pop off an element from the top of the stack. To read an element into the stack, the top elements must be popped off and are lost.
- A PDA is more powerful than FA. Any language which can be acceptable by FA can also be acceptable by PDA. PDA also accepts a class of language which even cannot be accepted by FA. Thus PDA is much more superior to FA.

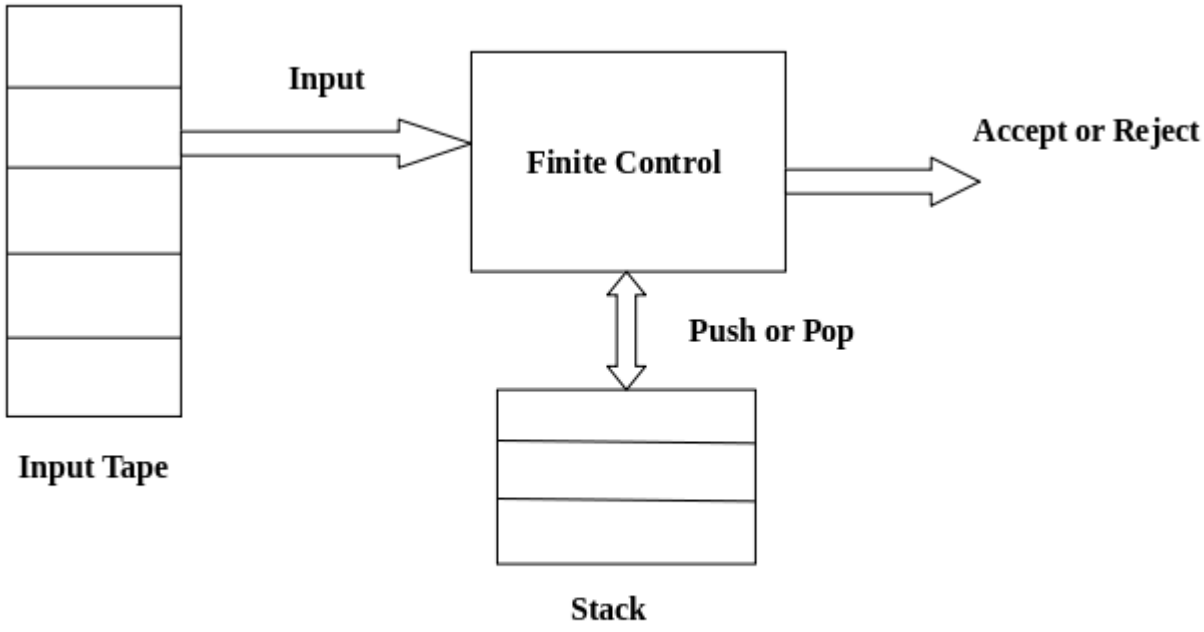


Fig: Pushdown Automata

PDA Components:

Input tape: The input tape is divided in many cells or symbols. The input head is read-only and may only move from left to right, one symbol at a time.

Finite control: The finite control has some pointer which points the current symbol which is to be read.

Stack: The stack is a structure in which we can push and remove the items from one end only. It has an infinite size. In PDA, the stack is used to store the items temporarily.

Formal definition of PDA:

The PDA can be defined as a collection of 7 components:

Q: the finite set of states

∑: the input set

Γ: a stack symbol which can be pushed and popped from the stack

q0: the initial state

Z: a start symbol which is in Γ.

F: a set of final states

δ: mapping function which is used for moving from current state to next state.

Instantaneous Description (ID)

ID is an informal notation of how a PDA computes an input string and make a decision that string is accepted or rejected.

An instantaneous description is a triple (q, w, α) where:

q describes the current state.

w describes the remaining input.

α describes the stack contents, top at the left.

Turnstile Notation:

⊢ sign describes the turnstile notation and represents one move.

⊢* sign describes a sequence of moves.

For example,

(p, b, T) ⊢ (q, w, α)

In the above example, while taking a transition from state p to q, the input symbol 'b' is consumed, and the top of the stack 'T' is represented by a new string α.

Example 1:

Design a PDA for accepting a language $\{a^n b^{2n} \mid n>=1\}$.

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 get popped from the stack.

The ID can be constructed as follows:

1. δ(q0, a, Z) = (q0, aaZ)
2. δ(q0, a, a) = (q0, aaa)

Now when we read b, we will change the state from q0 to q1 and start popping corresponding 'a'. Hence,

1. δ(q0, b, a) = (q1, ε)

Thus this process of popping 'b' will be repeated unless all the symbols are read. Note that popping action occurs in state q1 only.

1. δ(q1, b, a) = (q1, ε)

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

1. δ(q1, ε, Z) = (q2, ε)

Where

PDA = ({q0, q1, q2}, {a, b}, {a, Z}, δ, q0, Z, {q2})

We can summarize the ID as:

1. δ(q0, a, Z) = (q0, aaZ)

2. δ(q0, a, a) = (q0, aaa)
3. δ(q0, b, a) = (q1, ε)
4. δ(q1, b, a) = (q1, ε)
5. δ(q1, ε, Z) = (q2, ε)

Now we will simulate this PDA for the input string "aaabbbbbb".

1. δ(q0, aaabbbbbb, Z) ⊢ δ(q0, aabbbbbb, aaZ)
2. ⊢ δ(q0, abbbbbb, aaaaZ)
3. ⊢ δ(q0, bbbbbb, aaaaaaZ)
4. ⊢ δ(q1, bbbbb, aaaaaZ)
5. ⊢ δ(q1, bbbb, aaaaZ)
6. ⊢ δ(q1, bbb, aaaZ)
7. ⊢ δ(q1, bb, aaZ)
8. ⊢ δ(q1, b, aZ)
9. ⊢ δ(q1, ε, Z)
10. ⊢ δ(q2, ε)
11. ACCEPT

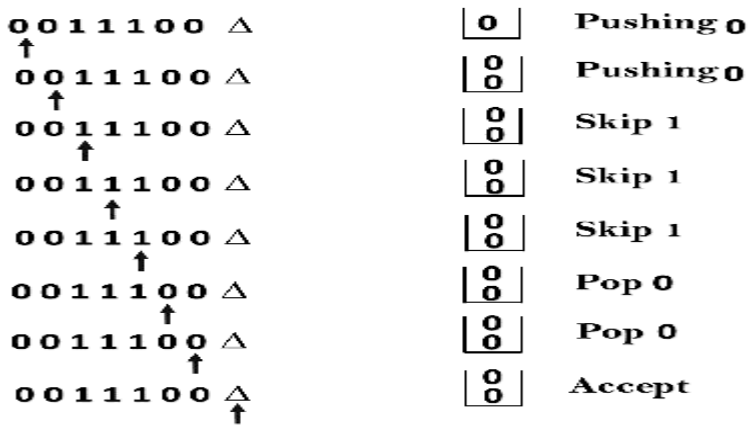
Example 2:

Design a PDA for accepting a language $\{0^n1^m0^n \mid m, n>=1\}$.

Solution: In this PDA, n number of 0's are followed by any number of 1's followed n number of 0's. Hence the logic for design of such PDA will be as follows:

Push all 0's onto the stack on encountering first 0's. 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:

1. $\delta(q0, 0, Z) = \delta(q0, 0Z)$
2. $\delta(q0, 0, 0) = \delta(q0, 00)$
3. $\delta(q0, 1, 0) = \delta(q1, 0)$
4. $\delta(q0, 1, 0) = \delta(q1, 0)$
5. $\delta(q1, 0, 0) = \delta(q1, \varepsilon)$
6. $\delta(q0, \varepsilon, Z) = \delta(q2, Z)$ (ACCEPT state)

Now we will simulate this PDA for the input string "0011100".

1. $\delta(q0, 0011100, Z) \vdash \delta(q0, 011100, 0Z)$
2. $\vdash \delta(q0, 11100, 00Z)$
3. $\vdash \delta(q0, 1100, 00Z)$
4. $\vdash \delta(q1, 100, 00Z)$
5. $\vdash \delta(q1, 00, 00Z)$
6. $\vdash \delta(q1, 0, 0Z)$
7. $\vdash \delta(q1, \varepsilon, Z)$
8. $\vdash \delta(q2, Z)$
9. ACCEPT

Problem: Design a non deterministic PDA for accepting the language $L = \{wwR\ w \in (a, b)^+\}$, i.e.,

$L = \{aa, bb, abba, aabbaa, abaaba, \ldots\ldots\}$

Explanation: In this type of input string, one input has more than one transition states, hence it is called non-deterministic PDA, and the input string contain any order of 'a' and 'b'. Each input alphabet has more than one possibility to move next state. And finally when the stack is empty then the string is accepted by the NPDA. In this NPDA we used some symbols which are given below:

$\Gamma = \{ a, b, z \}$

Where, Γ = set of all the stack alphabet
z = stack start symbol
a = input alphabet
b = input alphabet

The approach used in the construction of PDA –

As we want to design an NPDA, thus every times 'a' or 'b' comes then either push into the stack or move into the next state. It is dependent on a string. When we see the input alphabet which is equal to the top of the stack then that time pop operation applies on the stack and move to the next step. So, in the end, if the stack becomes empty then we can say that the string is accepted by the PDA.

STACK Transition Function

(q0, a, z) = (q0, az)

(q0, a, a) = (q0, aa)

(q0, b, z) = (q0, bz)

(q0, b, b) = (q0, bb)

(q0, a, b) = (q0, ab)

(q0, b, a) = (q0, ba)

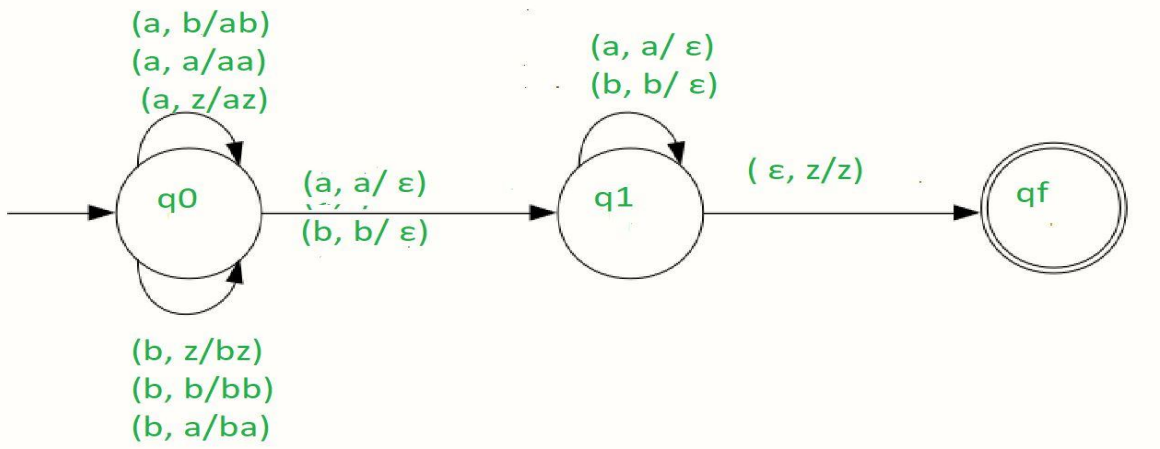
(q0, a, a) = (q1, ∈)

(q0, b, b) = (q1, ∈)

(q1, a, a) = (q1, ∈)

(q1, b, b) = (q1, ∈)

(q1, ∈, z) = (qf, z)



Required NPDA

So, this is our required non-deterministic PDA for accepting the language L = {wwR w ∈ (a, b)*}

Example:

We will take one input string: "abbbba".

- Scan string from left to right
- The first input is 'a' and follows the rule:
- on input 'a' and STACK alphabet Z, push the two 'a's into STACK as: (a, Z/aZ) and state will be q0
- on input 'b' and STACK alphabet 'a', push the 'b' into STACK as: (b, a/ba) and state will be q0

- on input ‘b’ and STACK alphabet ‘b’, push the ‘b’ into STACK as: (b, b/bb) and state will be q0
- on input ‘b’ and STACK alphabet ‘b’ (state is q1), pop one ‘b’ from STACK as: (b, b/∈) and state will be q1
- on input ‘b’ and STACK alphabet ‘b’ (state is q1), pop one ‘b’ from STACK as: (b, b/∈) and state will be q1
- on input ‘a’ and STACK alphabet ‘a’ and state q1, pop one ‘a’ from STACK as: (a, a/∈) and state will remain q1
- on input ∈ and STACK alphabet Z, go to the final state(qf) as : (∈, Z/Z)

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