1. What are Pushdown Automata, how it works and show with example?

Pushdown Automata is a finite automata with extra memory called stack which helps Pushdown automata to recognize Context Free Languages.

A Pushdown Automata (PDA) can be defined as :

- Q is the set of states
- ∑is the set of input symbols
- Γ is the set of pushdown symbols (which can be pushed and popped from stack)
- q0 is the initial state
- Z is the initial pushdown symbol (which is initially present in stack)
- F is the set of final states
- δ is a transition function which maps Q x {Σ ∪ ∈} x Γ into Q x Γ*. In a given state, PDA will read input symbol and stack symbol (top of the stack) and move to a new state and change the symbol of stack.

Example : Define the pushdown automata for language $\{a^n b^n \mid n > 0\}$

Solution : M = where Q = { q0, q1 } and Σ = { a, b } and Γ = { A, Z } and δ is given by :

δ(q0, a, Z) = { (q0, AZ) }
δ(q0, a, A) = { (q0, AA) }
δ(q0, b, A) = { (q1, ∈) }
δ(q1, b, A) = { (q1, ∈) }
δ(q1, ∈, Z) = { (q1, ∈) }

Let us see how this automata works for aaabbb

Row	State	Input	δ(transition function used)	Stack(Leftmost symbol represents top of stack)	State after move
1	q0	aaabbb		Z	q0
2	q0	$\underline{a}$aabbb	δ(q0,a,Z)={(q0,AZ)}	AZ	q0
3	q0	a$\underline{a}$abbb	δ(q0,a,A)={(q0,AA)}	AAZ	q0
4	q0	aa$\underline{a}$bbb	δ(q0,a,A)={(q0,AA)}	AAAZ	q0
5	q0	aaa$\underline{b}$bb	δ(q0,b,A)={(q1,∈)}	AAZ	q1
6	q1	aaab$\underline{b}$b	δ(q1,b,A)= {(q1,∈)}	AZ	q1
7	q1	aaabb$\underline{b}$	δ(q1,b,A)= {(q1,∈)}	Z	q1
8	q1	∈	δ(q1, ∈,Z)= {(q1,∈)}	∈	q1

Note :

- The above pushdown automaton is deterministic in nature because there is only one move from a state on an input symbol and stack symbol.
- The non-deterministic pushdown automata can have more than one move from a state on an input symbol and stack symbol.
- It is not always possible to convert non-deterministic pushdown automata to deterministic pushdown automata.
- Expressive Power of non-deterministic PDA is more as compared to expressive deterministic PDA as some languages which are accepted by NPDA but not by deterministic PDA which will be discussed in next article.
- The push down automata can either be implemented using accepetance by empty stack or accepetance by final state and one can be converted to another.

2. What are Pushdown Automata and Language, write clear about pushdown Automata and language?

The formal definition of the pushdown automata model is similar to that of nondeterministic finite automata, except that one must also specify the alphabet of stack symbols and alter the form of the transition function so that it specifies how the stack operates. Before we get to the definition, let us introduce some notation that will be useful for discussing stack

operations. For any alphabet Γ, which we will refer to as the stack alphabet, the stack operation alphabet ǀ Γ is defined as ǀ Γ = {↓, ↑} × Γ. (11.2) The alphabet ǀ Γ represents the possible stack operations for a stack that uses the alphabet Γ; for each a ∈ Γ we imagine that the symbol (↓, a) represents pushing a onto the stack, and that the symbol (↑, a) represents popping a off of the stack. Definition 11.1. A pushdown automaton (or PDA for short) is 6-tuple P = (Q, Σ, Γ, δ, q0, F) (11.3) where Q is a finite and nonempty set of states, Σ is an alphabet (called the input alphabet), Γ is an alphabet (called the stack alphabet), which must satisfy Σ∩ ǀ Γ = ∅, δ is a function of the form δ : Q × Σ ∪ ǀ Γ ∪ {ε} → P(Q), (11.4) q0 ∈ Q is the start state, and F ⊆ Q is a set of accept states. The way to interpret a transition function having the above form is that the set of possible labels on transitions is Σ ∪ ǀ Γ ∪ {ε}; we can either read a symbol a, push a symbol from Γ onto the stack, pop a symbol from Γ off of the stack, or take an ε-transition.

The language of a PDA

There are two ways to define the language because there are two notions of acceptance:

PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ ($L(P) \subseteq \Sigma^*$).

Acceptance by final state

$$L(P) = \{w \mid (q_0, w, Z_0) \vdash_P^* (q, \epsilon, \gamma) \wedge q \in F\}$$

That is the PDA accepts the word w if there is any sequence of IDs starting from (q_0, w, Z_0) and leading to (q, ϵ, γ), where $q \in F$ is one of the final states. Here it doesn't play a role what the contents of the stack are at the end.

In our example the PDA P_0 would accept 0110 because $(q_0, 0110, \#) \vdash_{P_0}^* (q_2, \epsilon, \epsilon)$ and $q_2 \in F$. Hence we conclude $0110 \in L(P_0)$.

On the other hand since there is no successful sequence of IDs starting with $(q_0, 0011, \#)$ we know that $0011 \notin L(P_0)$.

Acceptance by empty stack

$$L(P) = \{w \mid (q_0, w, Z_0) \vdash_P^* (q, \epsilon, \epsilon)\}$$

That is the PDA accepts the word w if there is any sequence of IDs starting from (q_0, w, Z_0) and leading to (q, ϵ, ϵ), in this case the final state plays no role.If we specify a PDA for acceptance by empty stack we will leave out the set of final states F and just use $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$.

Our example automaton P_0 also works if we leave out F and use acceptance by empty stack.

We can always turn a PDA which use one acceptance method into one which uses the other. Hence, both acceptance criteria specify the same class of languages

Example

Construct Pushdown automata for L = {$0^n1^m2^m3^n$ | m,n ≥ 0}

Prerequisite – Pushdown automata, Pushdown automata acceptance by final state

Pushdown automata (PDA) plays a significant role in compiler design. Therefore there is a need to have a good hands on PDA. Our aim is to construct a PDA for L = {$0^n1^m2^m3^n$ | m,n ≥ 0}

Examples –

```
Input : 00011112222333

Output : Accepted


Input : 0001122233

Output : Not Accepted
```

Approach used in this PDA –

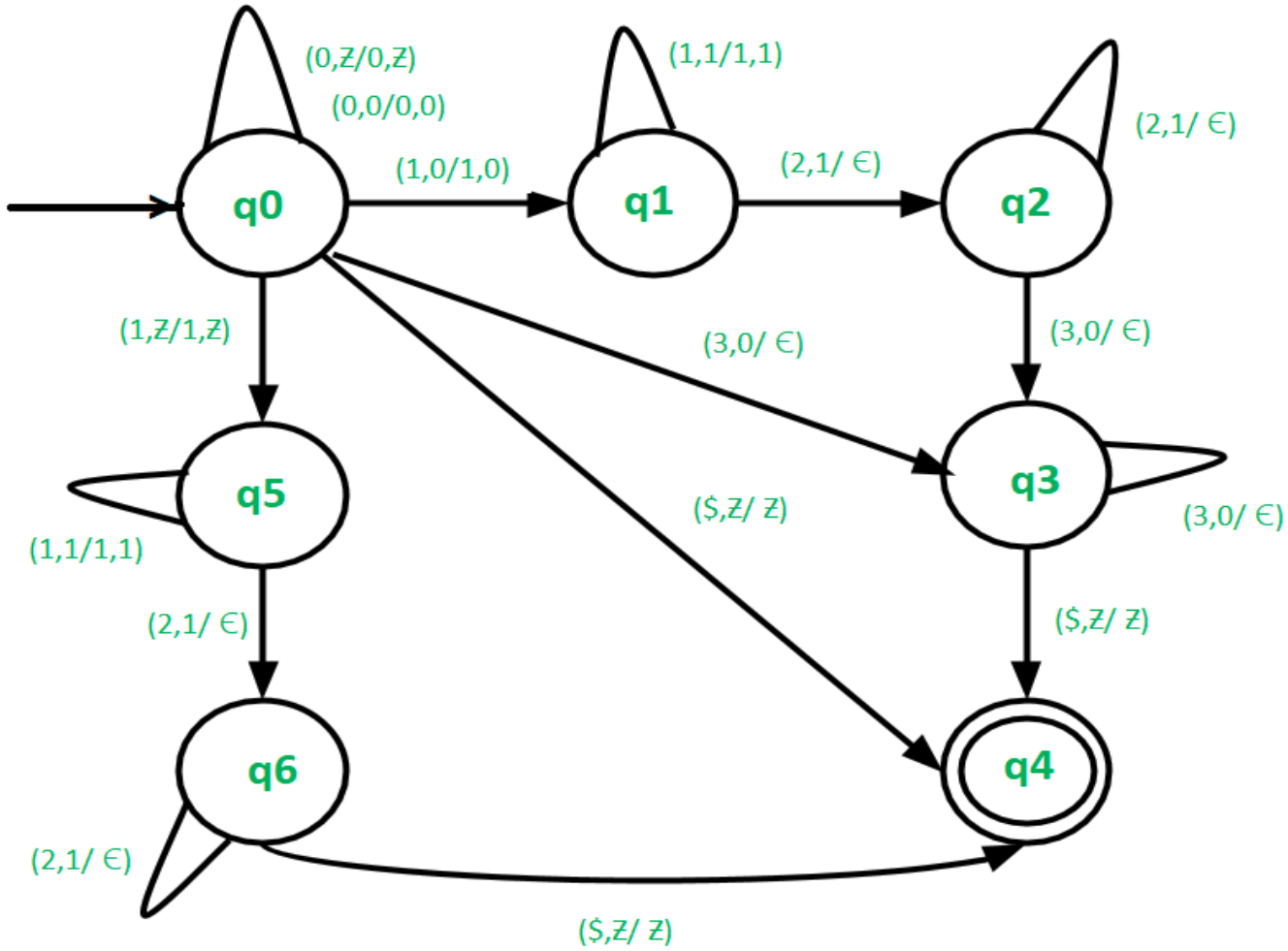
There can be 4 cases while processing the given input string.

- **Case-1: m = 0 –** In this cases the input string will be of the form {0^n3^n}. In this condition, keep on pushing 0's in the stack until we

encounter with 3. On receiving 3 check if top of stack is 0, then pop it (0) from the stack. Pop 0’s until all the 3’s of the input string are processed. If we reach to the end of input string and stack becomes empty, then reach to final state i.e. accept the input string else move to dead state.

- **Case-2: n = 0 –** In this cases the input string will be of the form $\{1^m2^m\}$. In this condition, keep on pushing 1’s in the stack until we encounter with 2. On receiving 2 check if top of stack is 1, then pop it (1) from the stack. Keep on pop 1’s until all the 2’s of the input string are processed. If we reach to the end of input string and stack becomes empty, then reach to final state i.e. accept the input string else move to dead state.
- **Case-3: m, n>0 –** In this cases the input string will be of the form $\{0^n1^m2^m3^n\}$. In this condition, keep on pushing 0’s and 1’s in the stack until we encounter with 2. On receiving 2 check if top of stack is 1, then pop it (1) from the stack. Keep on pop 1’s until all the 2’s of the input string are processed. Then on receiving 3 check if top of stack is 0, then pop it (0) from the stack. Pop 0’s until all the 3’s of the input string are processed. If we reach to the end of input string and stack becomes empty, then reach to final state i.e. accept the input string else move to dead state.
- **Case-4: m = 0, n = 0 –** In this case the input string will be empty. Therefore directly jump to final state.

Final Pushdown automata for the given language is:



3. What is Context Sensitive Language and show with different examples?

Context-sensitive Language: The language that can be defined by context-sensitive grammar is called CSL. Properties of CSL are :

- Union, intersection and concatenation of two context-sensitive languages is context-sensitive.
- Complement of a context-sensitive language is context-sensitive.

Example –
Consider the following CSG.
S → abc/aAbc
Ab → bA
Ac → Bbcc
bB → Bb
aB → aa/aaA
What is the language generated by this grammar?
Solution:
S → aAbc
→ abAc
→ abBbcc
→ aBbbcc
→ aaAbbcc
→ aabAbcc
→ aabbAcc
→ aabbBbccc
→ aabBbbccc
→ aaBbbbccc
→ aaabbbccc
The language generated by this grammar is {$\mathbf{a^n b^n c^n}$ | n≥1}.

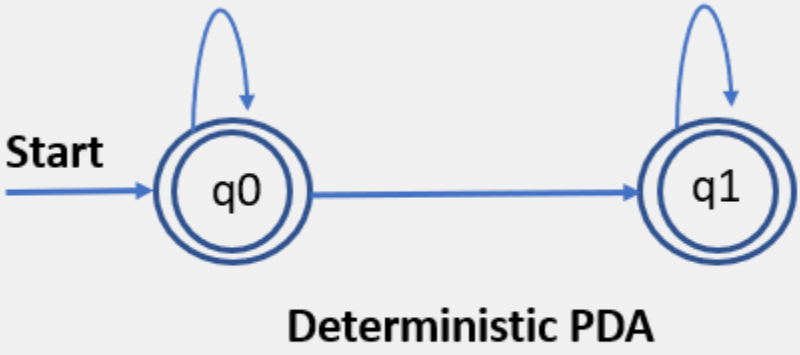
4. What is Deterministic and Nondeterministic Pushdown Automata mean express in examples for both?

A, Deterministic Pushdown Automata

The **Deterministic Pushdown Automata** is a variation of pushdown automata that accepts the deterministic context-free languages.
A language **L(A)** is accepted by a deterministic pushdown automata if and only if there is a single computation from the initial configuration until an accepting one for all strings belonging to **L(A)**. It is not as powerful as non-deterministic finite automata. That's why it is less in use and used only where determinism is much easier to implement.
A **PDA** is said to be deterministic if its transition function **δ(q,a,X)** has at most one member for -

```
a ∈ Σ U {ε}
```

So, for a deterministic PDA, there is at most one transition possible in any combination of state, input symbol and stack top.



Deterministic PDA

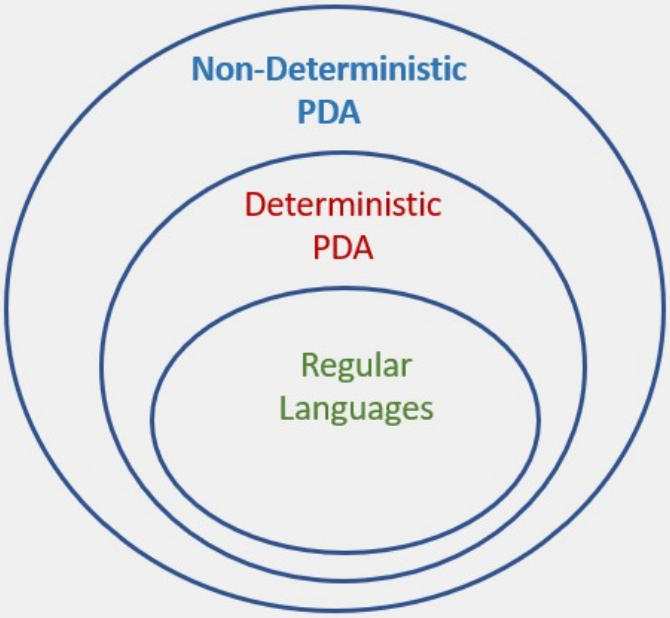
Formal Definition of Deterministic PDA

A Deterministic PDA is 5 tuple -

```
M = (Σ,Γ,Q,δ,q)
```

Σ - It is a finite set which does not contain a blank symbol,
Γ - a finite set of stack alphabet,
Q - set of states,
q - start state,
δ - a transition function, denoted as -

$\delta : Q \times (\Sigma \cup \{\square\}) \times \Gamma \to Q \times \{N,R\} \times \Gamma^*$



B, non-deterministic pushdown automaton

A *non-deterministic pushdown automaton* (NPDA), or just *pushdown automaton* (PDA) is a variation on the idea of a non-deterministic finite automaton (NDFA). Unlike an NDFA, a PDA is associated with a stack (hence the name *pushdown*). The transition function must also take into account the "state" of the stack.

Formally defined, a pushdown automaton M is a 7-tuple $M=(Q,\Sigma,\Gamma,T,q0,\bot,F)$, where $Q,\Sigma,q0$, and F, like those in an NDFA, are the set of states, the input alphabet, the start state, and the set of final states respectively. Γ is the *stack alphabet*, specifying the set of symbols that can be pushed onto the stack. Γ is not necessarily disjoint from Σ. $\bot$ is an element of Γ called the *start stack symbol*. The transition function is

T
:Q×(Σ∪{λ})×Γ→P(Q×Γ∗).T:Q×(Σ∪{λ})×Γ Γ*)
→𝒫(Q× .

How It Works

To see how the computing machine MM works, first imagine MM with the following features:

1. a finite set QQ of internal states,
2. a horizontal tape of cells each containing an input symbol of ΣΣ,
3. a tape reader that reads at most one tape cell in any given internal state, and
4. a vertical stack of cells storing symbols of ΓΓ.

Now, given that MM is in state pp, with symbol AA on top of the stack, and tape reader pointing at a tape cell containing symbol aa, it may do one of the following:

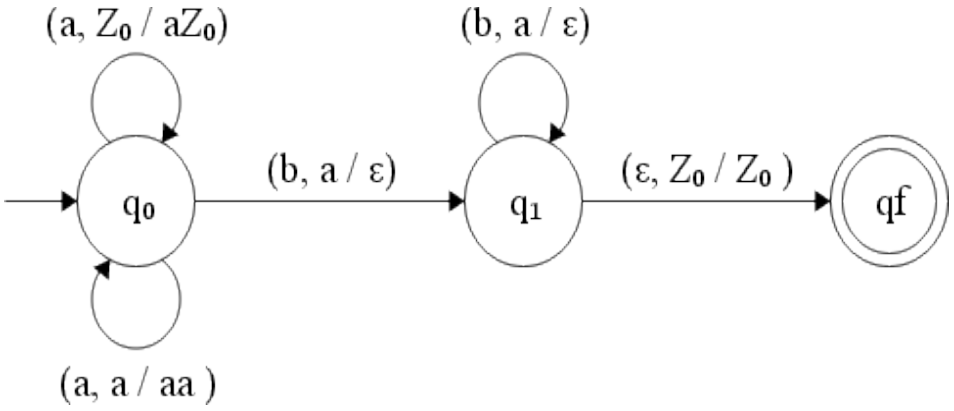
- if T(p,a,A)≠∅T(p,a,A)≠∅, then it
  1. (a) "pops" AA off the stack,
  2. (b) "pushes" word A1⋯AnA1⋯An onto the stack, by starting with symbol AnAn, and ending with symbol A1A1,
  3. (c) "consumes" aa by moving the tape reader to the right of the cell containing aa, and
  4. (d) enters state qq,

  provided that (q,A1⋯An)∈T(p,a,A)(q,A1⋯An)∈T(p,a,A);
  if T(p,a,A)=∅T(p,a,A)=∅, then MM does nothing.

- if T(p,λ,A)≠∅T(p,λ,A)≠∅, then, without reading aa, it
  1. (a) "pops" AA off the stack,
  2. (b) "pushes" word A1⋯AnA1⋯An onto the stack, and
  3. (c) enters state qq,

  as long as (q,A1⋯An)∈T(p,λ,A)(q,A1⋯An)∈T(p,λ,A);
  if T(p,λ,A)=∅T(p,λ,A)=∅, then MM does nothing.

If (q,λ)∈T(p,a,A)(q,λ)∈T(p,a,A), then AA gets popped off, and nothing gets pushed onto the stack.


(a, Z0 / aZ0)
(b, a / ε)
(b, a / ε)
(ε, Z0 / Z0)
q0
q1
qf
(a, a / aa)