PDA

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

Q: Finite set of states

Σ: input symbol

δ: Transition Function

q_0: Initial state

Z_0: Bottom of the stack

F: set of final states

Γ: stack alphabet (tow)

D $\delta : Q \times \{\Sigma \cup \in\} \times \Gamma \to Q \times \Gamma^*$

N $\delta : Q \times \{\Sigma \cup \in\} \times \Gamma \to 2^{(Q \times \Gamma^*)}$

The language which is generated by Context free Grammar is called context free language.

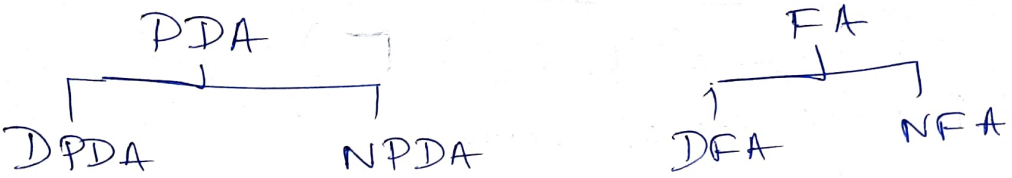
In order to accept a Regular language we need a FA. In order to accept a context free language we need a machine which is similar to FA and it is called push down Automata.

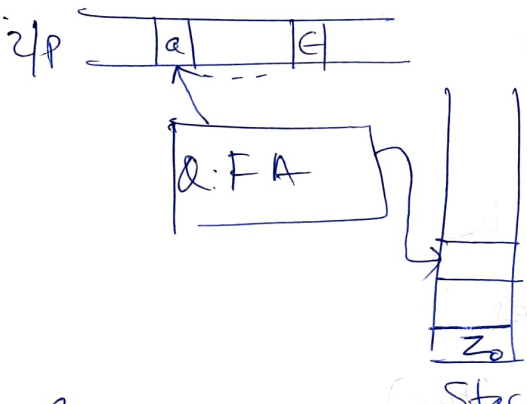
PDA is nothing but a FA to which some memory element is added and that memory element is stacked.

PDA - FA + stack

So if we add a stack to FA then we are going

to PDA.





$\delta : Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow Q \times \Gamma^*$

(Σ = i/p, Γ = Top of the stack, Q = s, Γ* = Top of the stack)

If you see deterministic PDA, the transition function will be like this.

If you are in a state and if you see some input alphabet Σ or ∈ and if you see Γ (one of the symbol from the top of the stack), depending on these three, it should decide to go to (one state) some state Q (same state or some other state) then something Print or pushing on the top of the stack.

Then this is called Deterministic PDA.

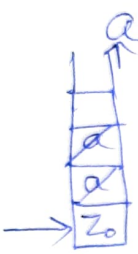
Non deterministic PDA

$\delta : Q \times \{\Sigma \cup \epsilon\} \times \Gamma \rightarrow 2^{(Q \times \Gamma^*)}$

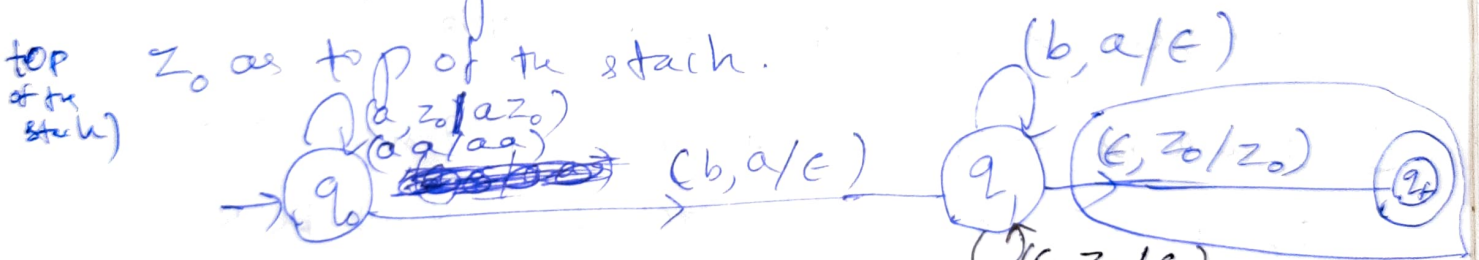
If you are in some state Q and if you will be seeing some input and there will be some top of the stack which means in one state seeing

some input and seeing the top of the stack, if you decide to be want to go more than one state and then print more than one symbol on top of the stack then it is called nondeterministic PDA.

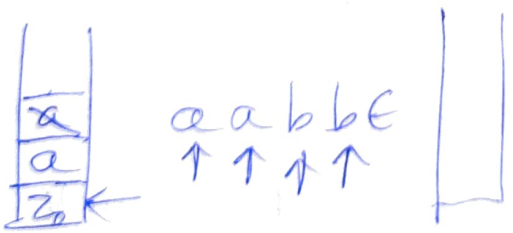
Ex Take a language $a^n b^n / n \geq 1$ which we want to accept



As long as you are getting a, you are pushing a on to the stack and as long as you seeing b, you keep on popping a and finally the input is over. How do you know that input is over. bcoz there will be ϵ and Z_0 as top of the stack.



Representing PDA through transition diagram



Initially as long as you see a input and Z_0 as top of the stack, push a then there will be a on the top of the stack and keep on pushing a. Then b is the input and a is on the top of the stack; I am simply popping out a. Then as long as I see b in the input,

and a is on the top of the stack I keep on popping them. Then finally the entire input is over. Z_0 is on the top of the stack and accepted. This is called state transition diagram of the PDA.

Transition function

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

(state, input, top of the stack)

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

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

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

$\delta(q_1, \epsilon, Z_0) = (q_f, Z_0)$ or (q_1, ϵ)

Representing PDA through transition function.

(1) I will be in state q_0 and then if I see a a and Z_0 is on the top of the stack then I will be in the same state q_0 and I will push aZ_0.

(2) Then I will be in state q_0, a is on the input and a is on the top of the stack then I will be on the same state q_0 and I am going to push aa.

(3) I will be in state q_0 and b is in the input and a is on the top of the stack, then I am going to the state q_1 (as b is input) and input is going to popped up.

(4) I will be in state q_1, I will get b and a will be on top of the stack then I will be in the same state q_1 and I will keep on popping a for every b.

(5) finally I will be in state q_1 and I will see the input ϵ, which means the entire input is over and Z_0 is on bottom of the stack then I am going to state q_f and I am going to leave it at Z_0. Acceptance by final state

(6) one other way of accepting it is, you need not use a final state. what we could do is whenever you see ϵ and Z_0 is on the top of the stack then you can leave it at ϵ. This is called Acceptance by empty stack.

PDA acceptance is of 2 types.

PDA
- Acceptance by final state
- Acceptance by Empty stack.

Note Deterministic and Nondeterministic PDA both are equivalent in power.

Q $w/n_a(w) = n_b(w)$

I want to accept set of all strings such that no. of a's in w is equal to no. of b's in w.

This problem is different than $a^n b^n / n \geq 1$

bo2

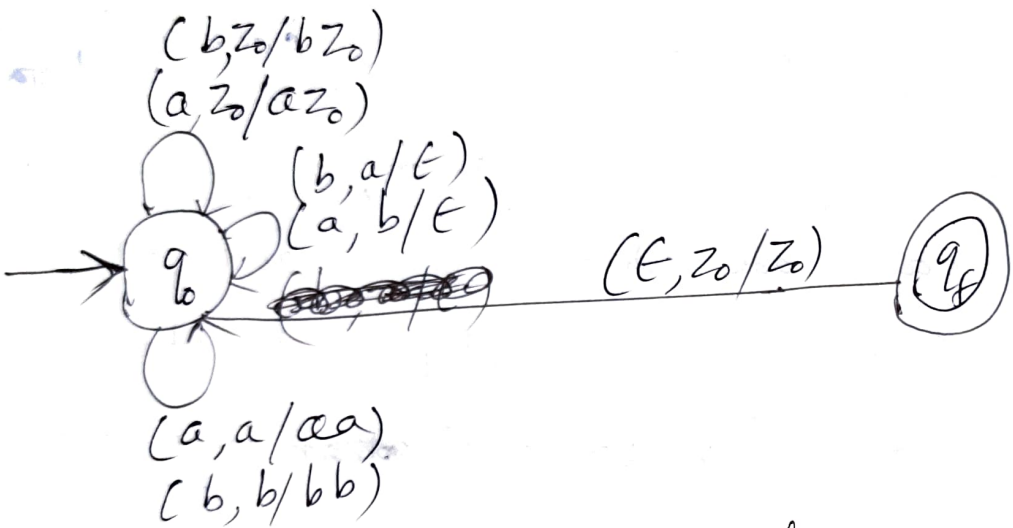
in $W / n_a(w) = n_b(w)$

the orders of a and b can be

ab bbaa abab abbq

aabb baba

aabb



Initially anything can come. like a can come or b can come. So we should be prepared for both of them. If Initially input is a and top of the stack is Z0 then we are going to push a.

or if initially b comes and top of the stack is Z0, we are going to push b.

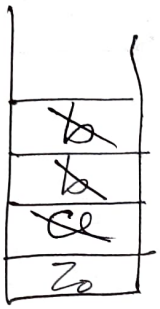
If input is a and a is on the top of the stack we are saving it.

If input is b and b is on the top of the stack then we are saving it.

If a is the input and b is on the top of the stack then we are going to pop up.

If b is in the input and a is on top of the stack then we are going to pop up.

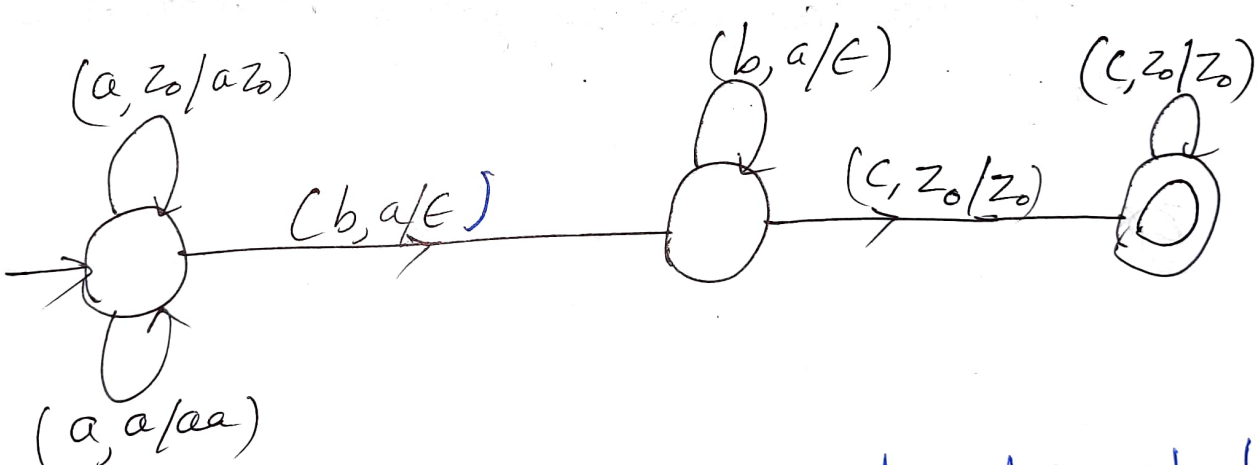
abbbaaϵ



In the previous question after a b comes. so we change the state. here after a, b comes and again a comes. So we need not to change the state - we can do it in one state.

Ex $a^n b^n c^m / n, m \geq 1$

Construct the PDA.



a's and b's have to be compared and they should be equal. for every a, we are going to push a and for every b we are going to pop a. I need not count c. bcz in c it will be accepted.