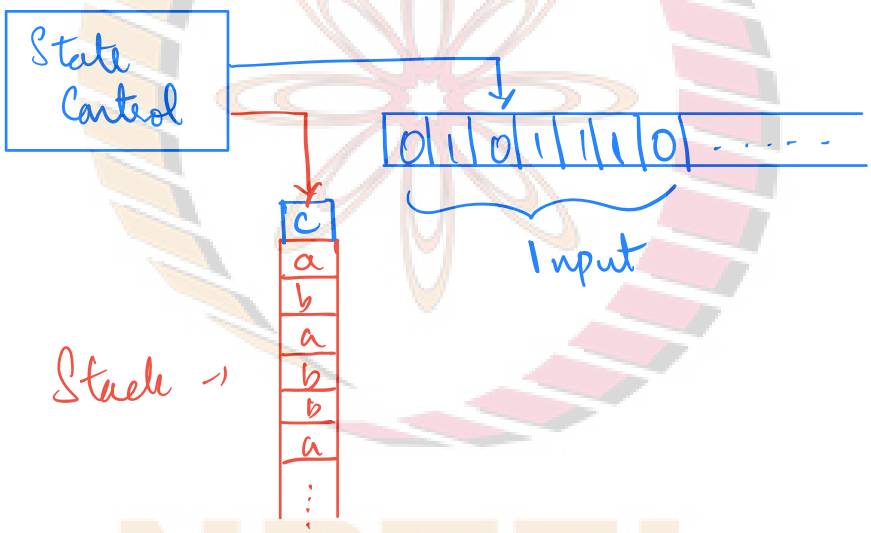
Pushdown Automata (PDA)

(Nondeterministic PDA).

This is different in terms of computation power from det. PDA.

→ Like NFA, but with a stack for computation

→ Stack : Simple unbounded memory but with restricted access.



In addition to moving between the states, the PDA can also choose to push | pop symbols to | from the stack. The state control also moves based on the stack symbols.

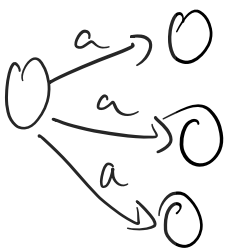
Def 2.13: A **pushdown automaton** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where Q, Σ, Γ and F are finite sets.

1. Q is the set of states
2. Σ is the input alphabet
3. Γ is the stack alphabet

} Finite Sets

4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function

In NFA, we had $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$



5. $q_0 \in Q$ is the start state
6. $F \subseteq Q$ is the set of accepting states.

The PDA M computes as follows. It **accepts** the input w, if $w = w_1 w_2 \ldots w_m$ where $w_i \in \Sigma_\varepsilon$, and a sequence of states $r_0, r_1,$

$r_2, \ldots r_m \in Q$ and $s_0, s_1, s_2, \ldots s_m \in \Gamma^*$ exist, where:

↑ Different from NFA's.

(1) $r_0 = q_0$ and $s_0 = \varepsilon$

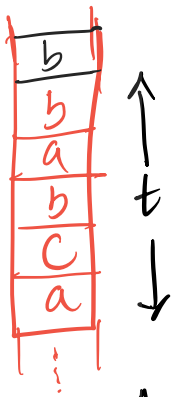
↑ Start State

↑ Stack empty

(2) For $i = 0, 1, \ldots m-1$, we have

$(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$

where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$



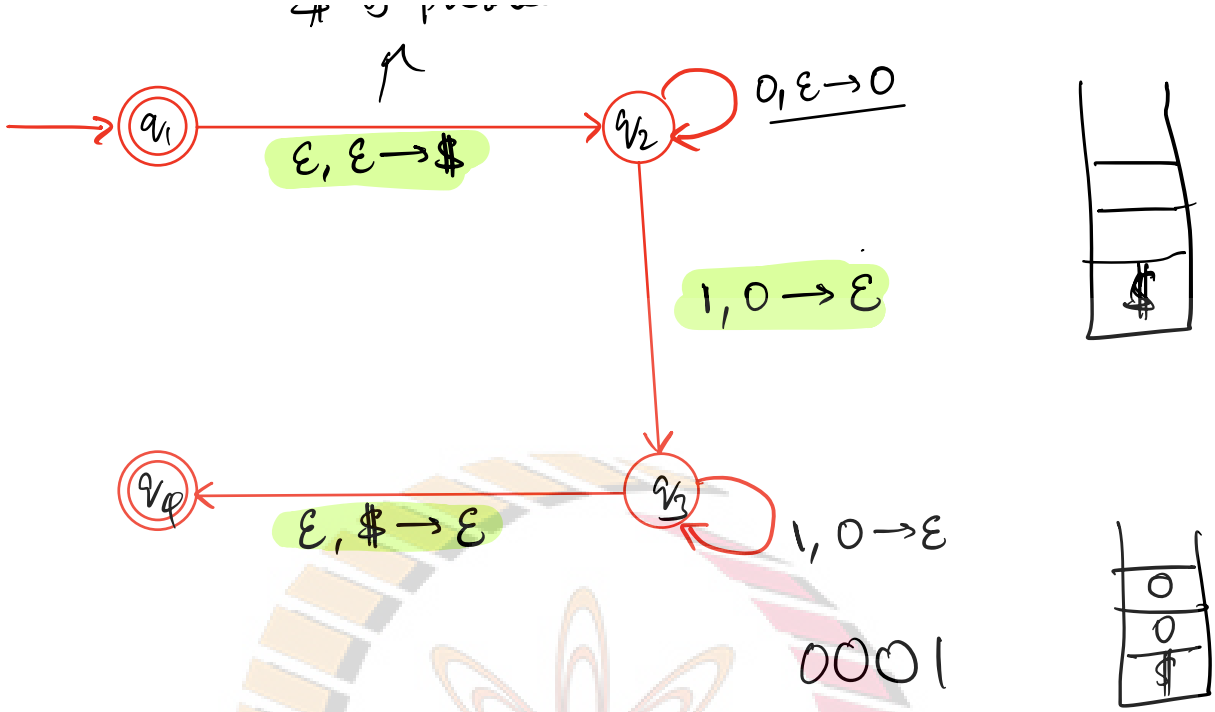
$t = babca$

(3) $r_m \in F$ (Stack need not be empty i.e., s_m need not be ε)

$a, b \to c$: Read a from the input
Read b from stack
Write c into stack

0001111

$ is pushed



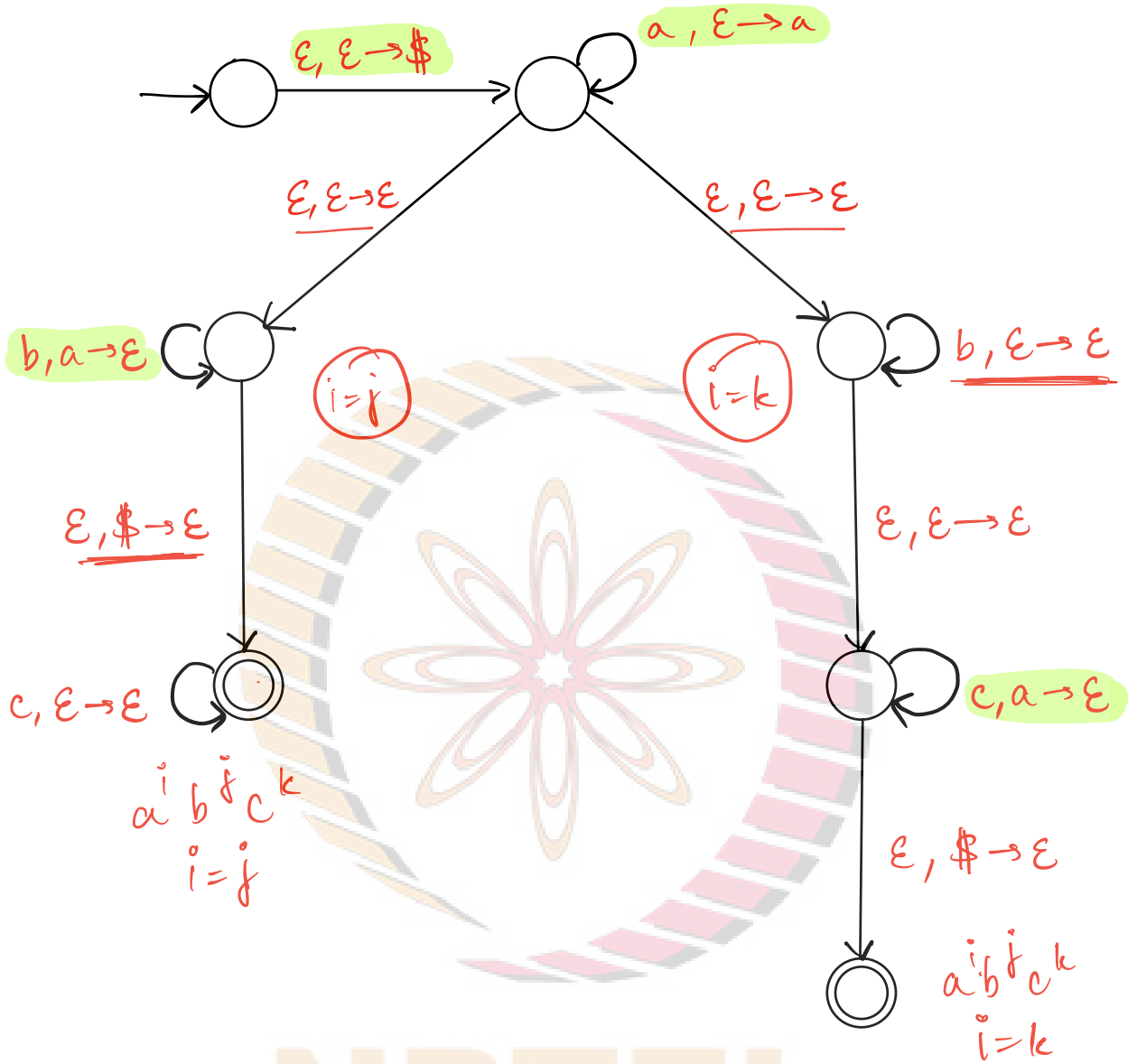
If this PDA has to accept, the string should be of the form $0^n 1^n$.

Here we use the \$ symbol to check if the stack is empty. The formal definition of PDA has no means to check if the stack is empty.

Exercise: Construct a PDA for all the strings that constitute properly nested parentheses.

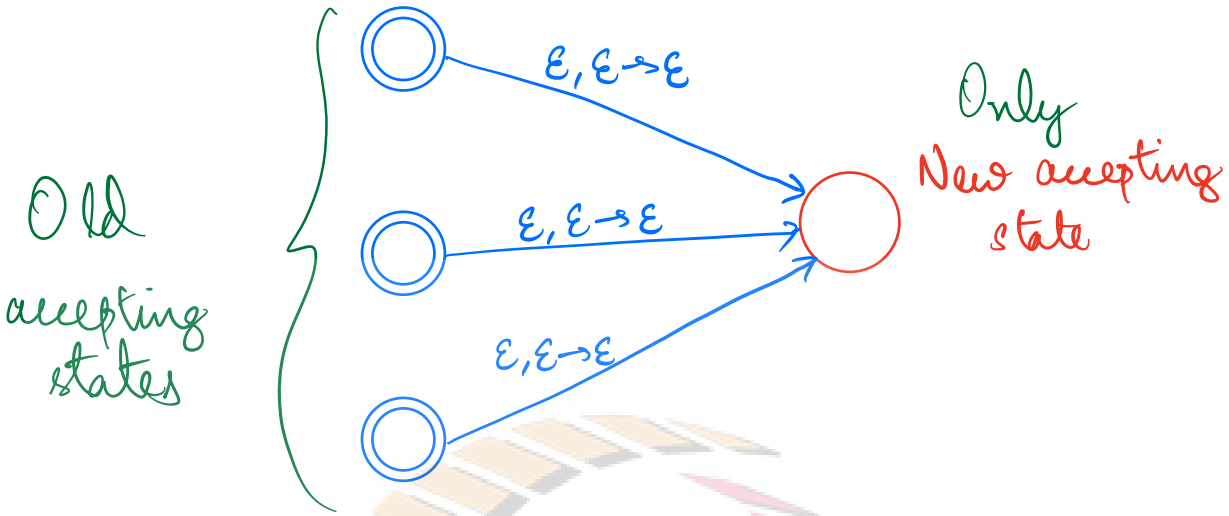
(( )) , ( )( )(( )) , (( )( ))(( ))

$L = \{ a^i b^j c^k \mid i, j, k \geq 0, i = j \text{ or } i = k \}$



Some Normalizations

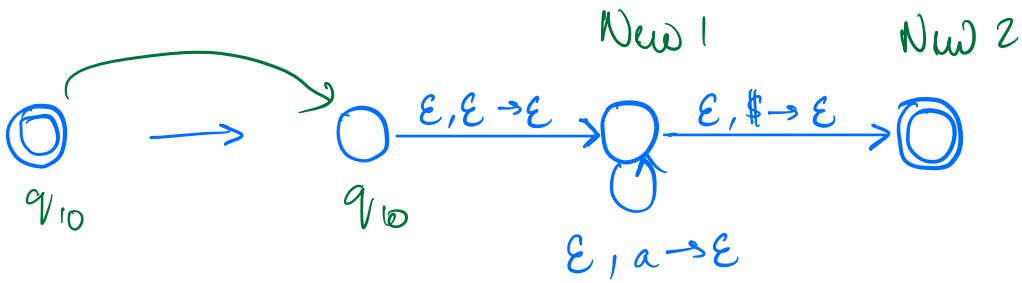
1. We can convert any PDA into a PDA with a single accepting state.



2. We can convert any PDA into an equivalent PDA that empties stack before accepting.

1. Put $ into stack initially

2. Empty all symbols after original accept. Accept only when $ is popped out.



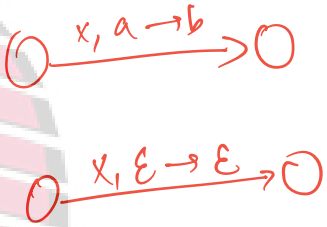
$\forall a \in \Gamma$

3. We can convert each PDA into a PDA where each transition pushes or pops, but not both.

We have 4 types of transitions

1. Push only.
2. Pop only.
3. Both push and pop
4. Neither push nor pop

} Exercise.



Intersection of a regular language and a CFL.

We saw that CFL's are not closed under intersection.

Theorem: If A is a regular language and B is a CFL, then $A \cap B$ is a CFL as well.

Proof Sketch: We can construct a PDA for $A \cap B$. We have a DFA M such that $L(M) = A$.

We have PDA P such that $L(P) = B$.

We use the Cartesian product idea (originally used to show that regular languages are closed under union).

Let $M = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and

$P = (Q_2, \Sigma, \Gamma, \delta_2, q_2, F_2)$.

We will construct PDA P':

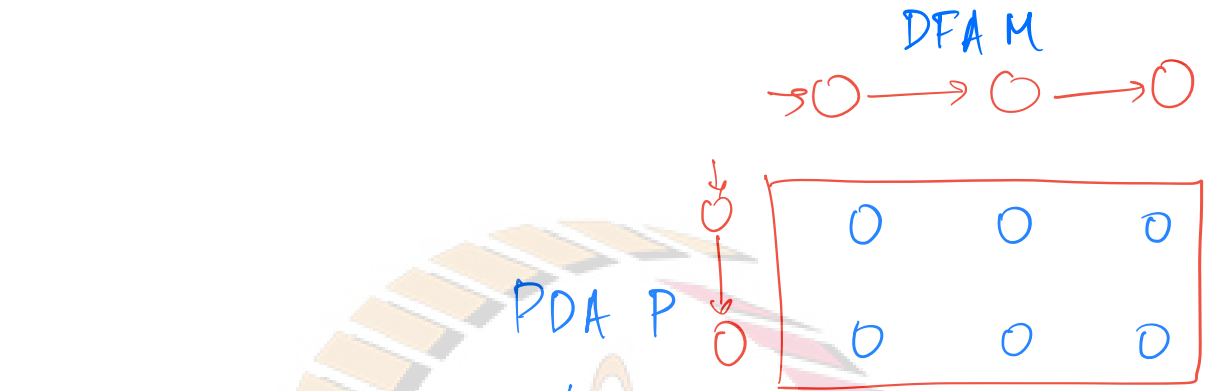
→ States of $P' = Q_1 \times Q_2$ (x_1, x_2)

→ Stack of P' = Stack of P.

→ Transition function of P' combines the transitions in M and P.

→ Accepting states of $P' = F_1 \times F_2$.

Exercise: Work out the details.



This resulting automaton will be a PDA.