In compilers, we use deterministic PDAs.

Def

A deterministic PDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where $Q, \Sigma, \Gamma$, and $F$ are finite sets, and

1. $Q$ is the set of states
2. $\Sigma$ is the input alphabet.
3. $\Gamma$ is the stack alphabet.
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow (Q \times \Gamma_\varepsilon) \cup \{\emptyset\}$ is the transition function.
5. $q_0 \in Q$ : start state
6. $F \subseteq Q$ ; set of accept states.

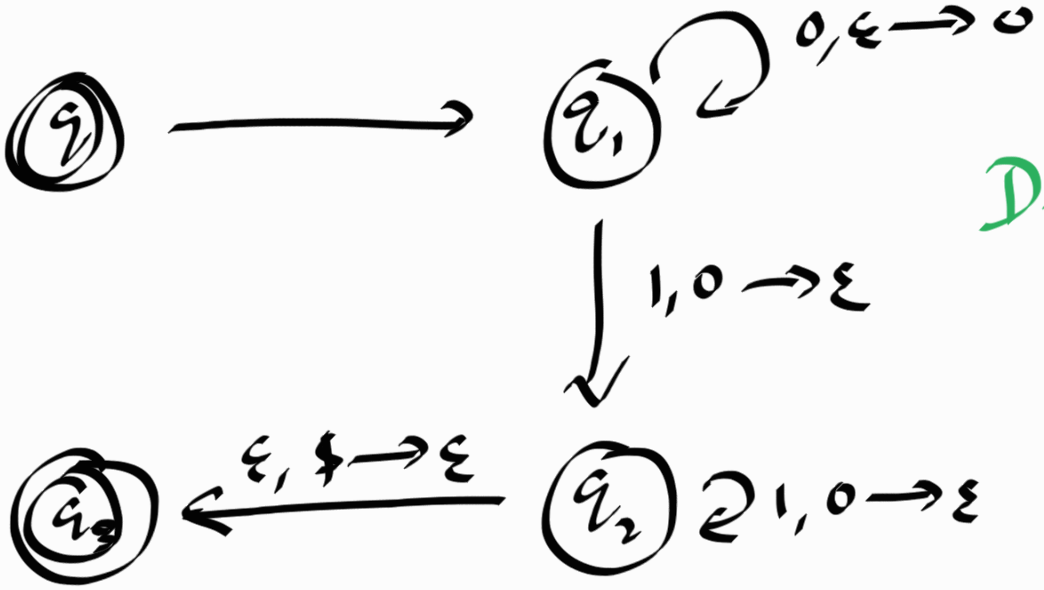
We have an additional constraint.

For every $q \in Q$, $a \in \Sigma$, and $x \in \Gamma$ exacty one of these values $\delta(q, a, x)$, $\delta(q, a, \varepsilon)$, $\delta(q, \varepsilon, x)$ and $\delta(q, \varepsilon, \varepsilon)$ is <u>not</u> $\emptyset$. (not non-empty)

# Example

PDA for $\{0^n 1^n \mid n \geq 0\}$

①




Deterministic

②




Non-deterministic

PDAs that we talked about in earlier lectures are non-deterministic PDAs.

They are equivalent to CFLs in their power.

But deterministic PDAs are not equivalent in power to CFLs.

Languages recognized by deterministic PDAs are called deterministic context free languages.

# Open problems in CFL.

Given a string $S$ of length $n$, compute/find the smallest grammar generating $S$.

Ex: $S = a^n b^n$, $|S| = 2n$

$$T \longrightarrow \underline{aaa \ldots a\,bbb \ldots b}$$

$$T \longrightarrow aTb \mid \varepsilon$$

Problem is can we do this efficiently?

End of mid-point of the semester