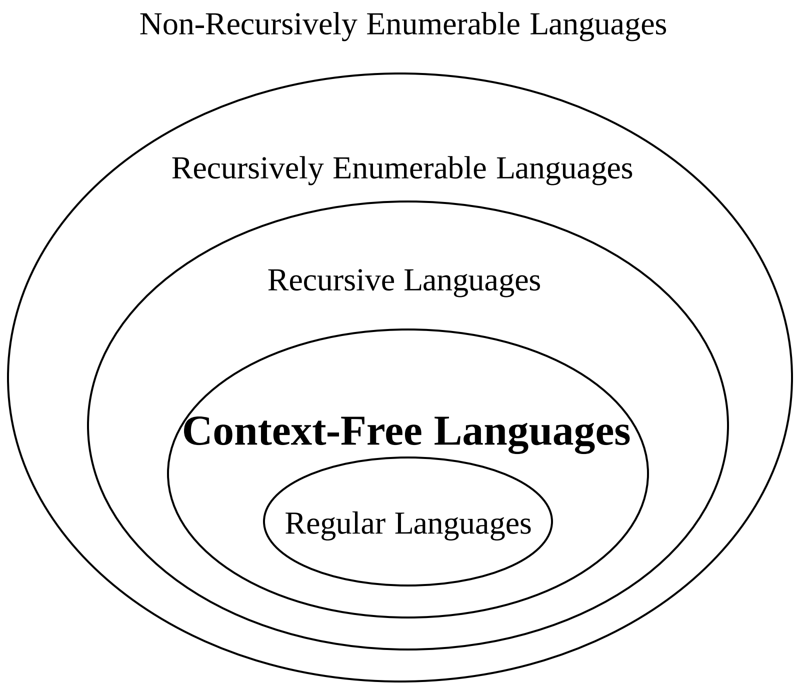
Pushdown Automata

Hierarchy of languages

Regular Languages ➔ Finite State Machines, Regular Expression

Context Free Languages ➔ Context Free Grammar, **Push-down Automata**



Pushdown Automata (PDA)

- **Informally:**
  - A PDA is an NFA-ε with a stack.
  - Transitions are modified to accommodate stack operations.
- **Questions:**
  - What is a stack?
  - How does a stack help?
- A DFA can "remember" only a finite amount of information, whereas a PDA can "remember" an infinite amount of (certain types of) information, in one memory-stack

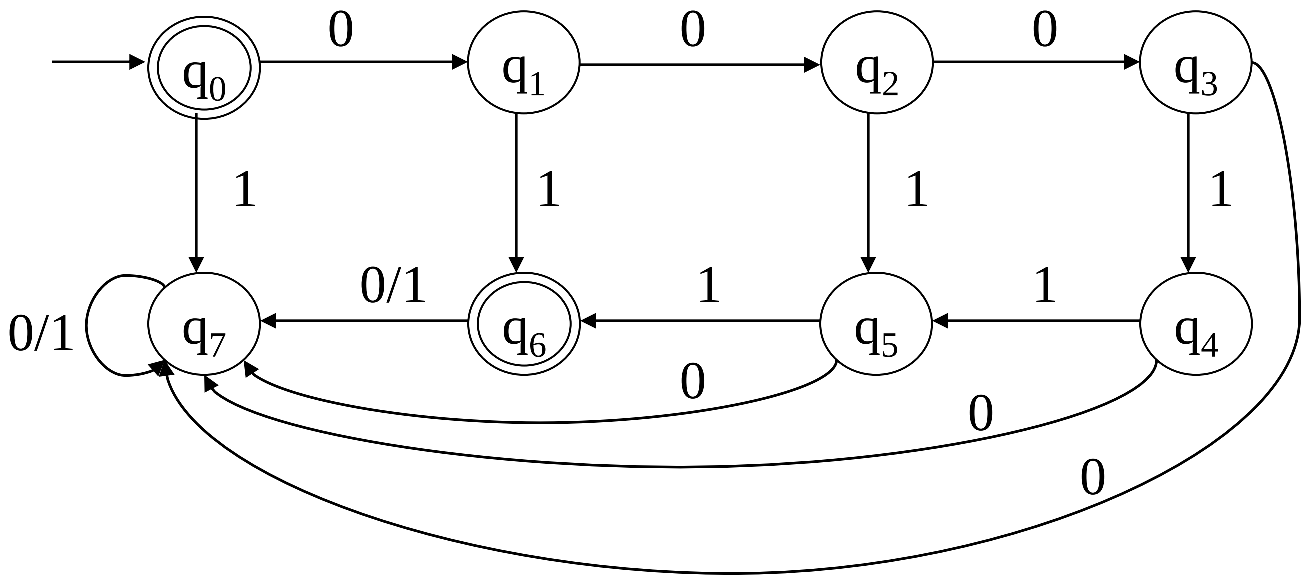
- **Example:**

  $\{0^n1^n \mid 0 =< n\}$ is *not* regular, but

  $\{0^n1^n \mid 0 \le n \le k$, for some fixed $k\}$ is regular, for any fixed k.

- **For k=3:**

  $L = \{\varepsilon, 01, 0011, 000111\}$



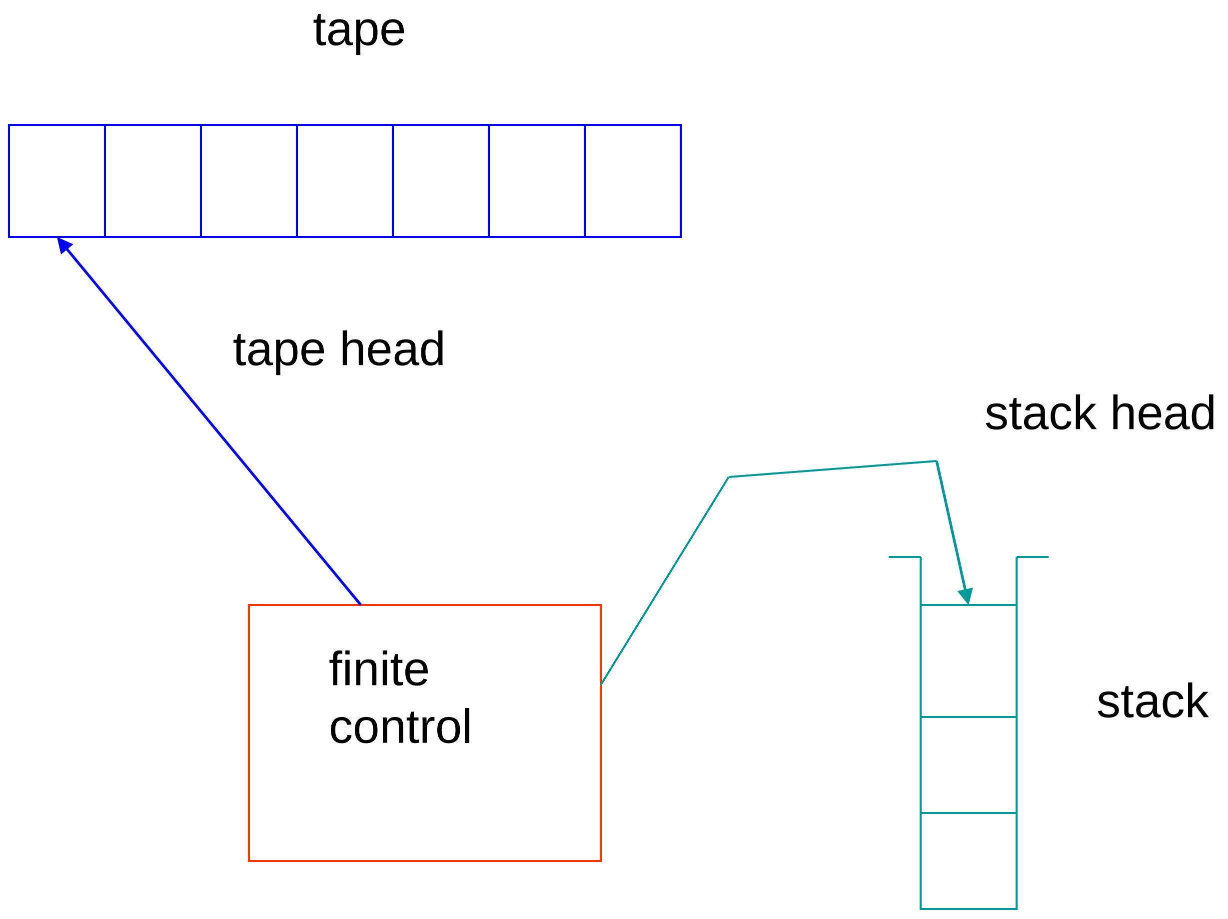
- In a DFA, each state remembers a finite amount of information.

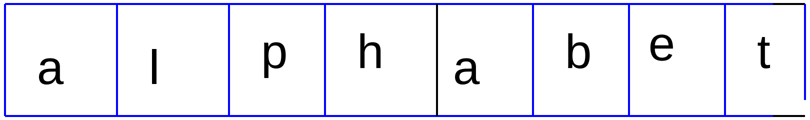
- To get $\{0^n1^n \mid 0 \leq n\}$ with a DFA would require an infinite number of states using the preceding technique.

- An infinite stack solves the problem for $\{0^n1^n \mid 0 \leq n\}$ as follows:
  - Read all 0's and place them on a stack
  - Read all 1's and match with the corresponding 0's on the stack

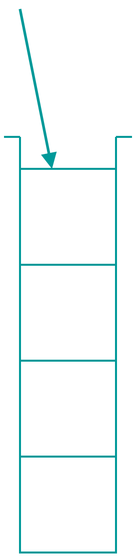
- Only need two states to do this in a PDA

- Similarly for $\{0^n1^m0^{n+m} \mid n,m \geq 0\}$


tape
tape head
stack head
finite
control
stack



The tape is divided into finitely many cells. Each cell contains a symbol in an alphabet Σ.

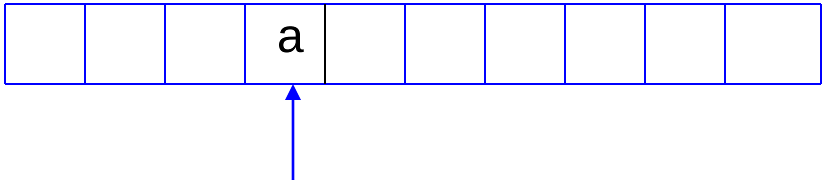


The stack head always scans the top symbol of the stack. It performs two basic operations:

Push: add a new symbol at the top.

Pop: read and remove the top symbol.

Alphabet of stack symbols: Γ



- The head scans at a cell on the tape and can *read* a symbol on the cell. In each move, the head can move to the right cell.

PDA: Example

Top Plate	State	0	1	C
Blue	q_1	Add a Blue plate and remain in state q_1	Add Green plate and remain in state q_1	Go to state q_2
	q_2	Remove Blue plate and remain in state q_2	-	-
Green	q_1	Add a Blue plate and remain in state q_1	Add Green plate and remain in state q_1	Go to state q_2
	q_2	-	Remove Green plate and remain in state q_2	-
Red	q_1	Add a Blue plate and remain in state q_1	Add Green plate and remain in state q_1	Go to state q_2
	q_2	Without waiting for input remove Red plate		

Formal Definition of a PDA

- A pushdown automaton (PDA) is a seven-tuple:

  $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

Q	A finite set of states
Σ	A finite input alphabet
Γ	A finite stack alphabet
q_0	The initial/starting state, q_0 is in Q
z_0	A starting stack symbol, is in Γ // need not always remain at the bottom of stack
F	A set of final/accepting states, which is a subset of Q
δ	A transition function, where

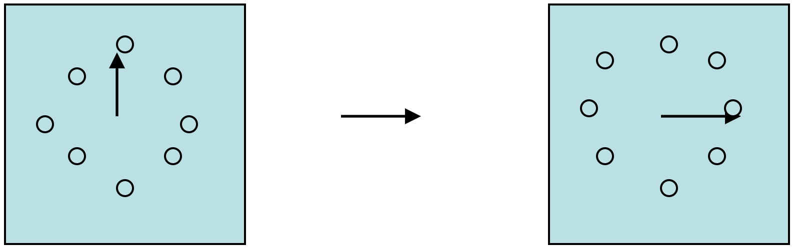
$$\delta: Q \text{ x } (\Sigma \cup \{\varepsilon\}) \text{ x } \Gamma \rightarrow \text{finite subsets of } Q \text{ x } \Gamma^*$$

- Consider the various parts of δ:

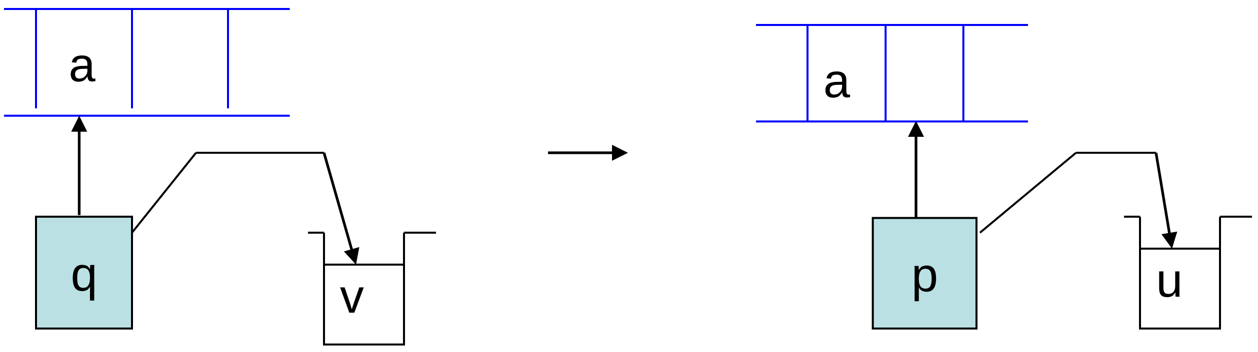
  $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow$ finite subsets of $Q \times \Gamma^*$

  – Q on the LHS means that at each step in a computation, a PDA must consider its' current state.
  – Γ on the LHS means that at each step in a computation, a PDA must consider the symbol on top of its' stack.
  – $\Sigma \cup \{\varepsilon\}$ on the LHS means that at each step in a computation, a PDA may or may not consider the current input symbol, i.e., it may have epsilon transitions.

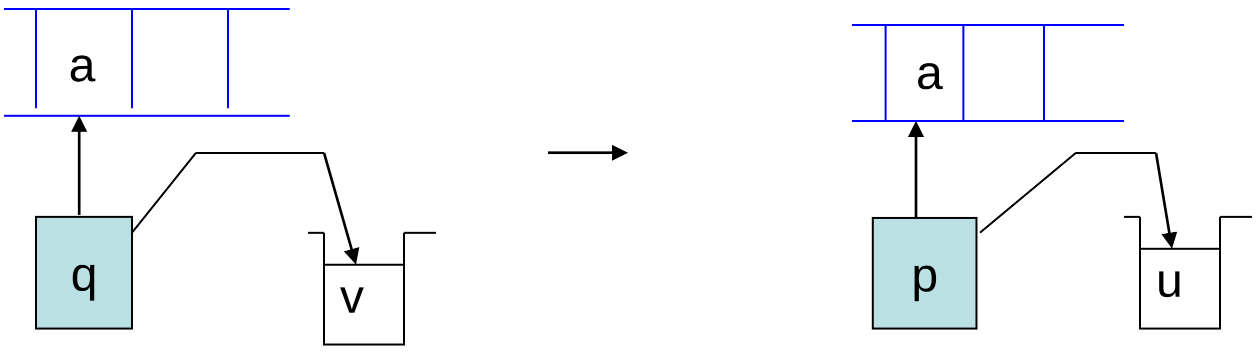
  – "Finite subsets" on the RHS means that at each step in a computation, a PDA may have several options.
  – Q on the RHS means that each option specifies a new state.
  – Γ^* on the RHS means that each option specifies zero or more stack symbols that will replace the top stack symbol, but *in a specific sequence*.



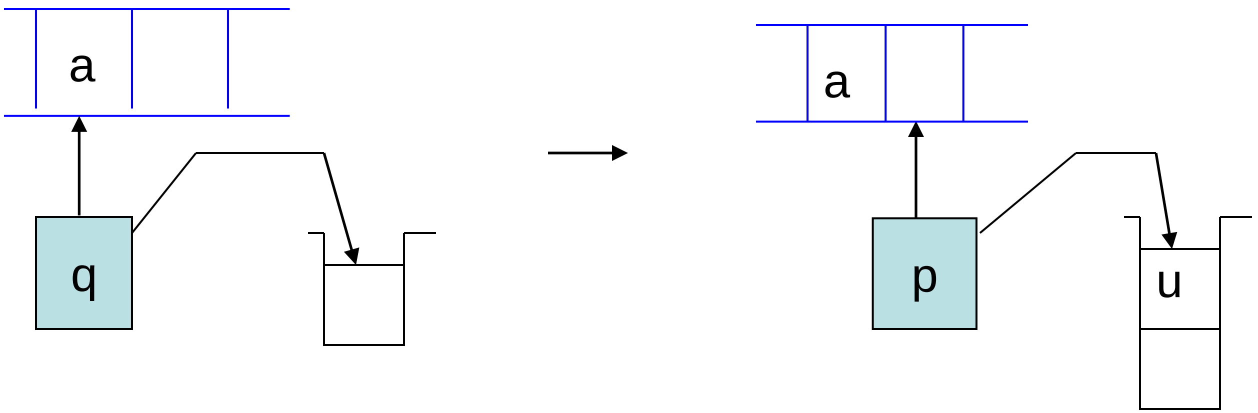
- The finite control has finitely many states which form a set Q. For each move, the state is changed according to the evaluation of a *transition function* $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow 2^{Q \times (\Gamma \cup \{\varepsilon\})}$.



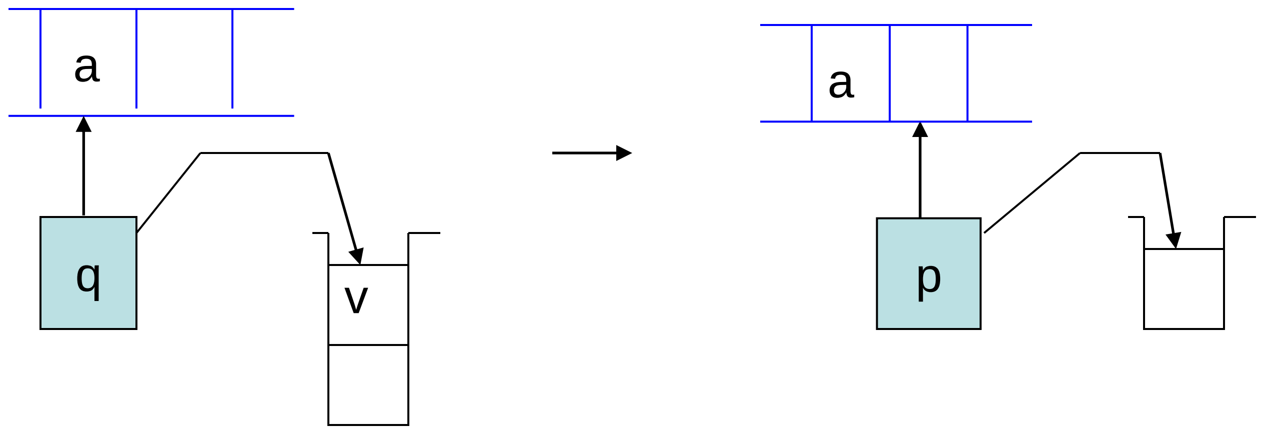
- *(p, u)* $\in$ *δ(q, a, v)* means that if the tape head reads *a,* the stack head read *v*, and the finite control is in the state *q*, then one of possible moves is that the next state is *p*, *v* is replaced by *u* at stack, and the tape head moves one cell to the right.



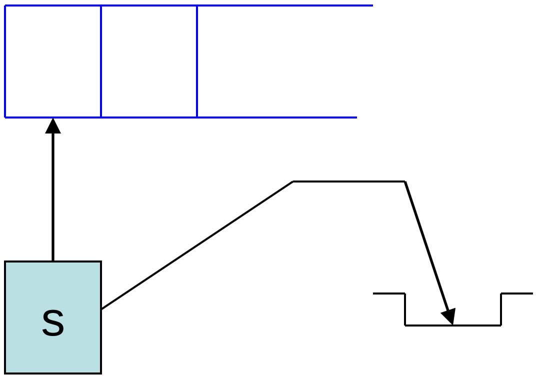
- *(p, u)* ∈ *δ(q, ε, v)* means that this a ε-move.



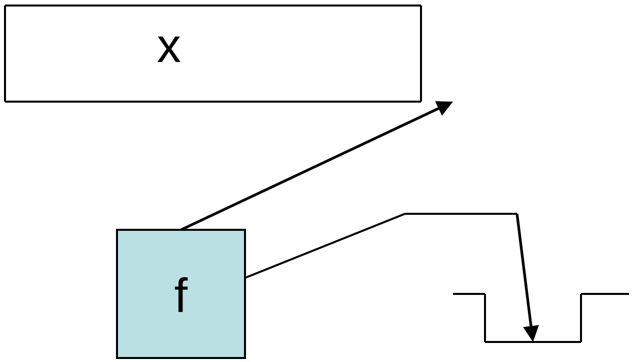
- *(p, u)* ∈ *δ(q, a, ε)* means that a push operation performs at stack.



- $(p, \varepsilon) \in \delta(q, a, v)$ means that a pop operation performs at stack



- There are some special states: an initial state s and a final set F of final states.
- Initially, the PDA is in the initial state s and the head scans the leftmost cell. The tape holds an input string. The stack is empty.

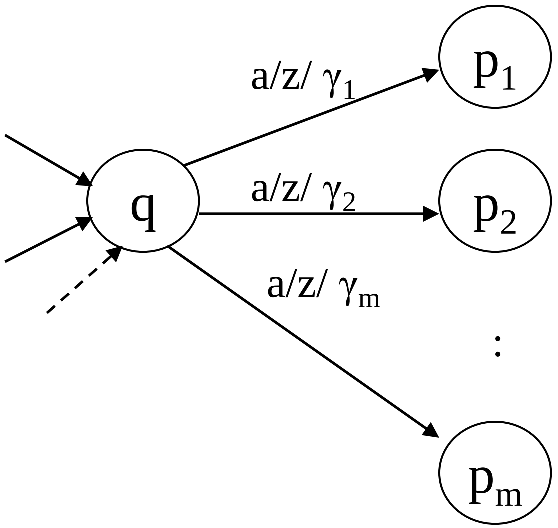


- When the head gets off the tape, the PDA stops. An input string *x* is accepted by the PDA if the PDA stops at a final state and the stack is empty.
- Otherwise, the input string is rejected.

- **Two types of PDA transitions:**

  $\delta(q, a, z) = \{(p_1,\gamma_1), (p_2,\gamma_2),\ldots, (p_m,\gamma_m)\}$

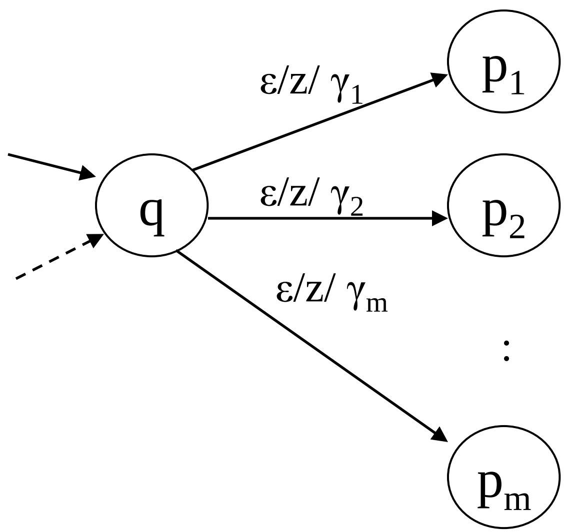
  - Current state is q
  - Current input symbol is a
  - Symbol currently on top of the stack z
  - Move to state p_i from q
  - Replace z with γ_i on the stack (leftmost symbol on top)
  - Move the input head to the next input symbol



- **Two types of PDA transitions:**

  $\delta(q, \varepsilon, z) = \{(p_1,\gamma_1), (p_2,\gamma_2),\ldots, (p_m,\gamma_m)\}$

  - Current state is q
  - Current input symbol is not considered
  - Symbol currently on top of the stack z
  - Move to state p_i from q
  - Replace z with γ_i on the stack (leftmost symbol on top)
  - **No input symbol is read**



PDA: Example

Top Plate	State	0	1	C
Blue	q_1	Add a Blue plate and remain in state q_1	Add Green plate and remain in state q_1	Go to state q_2
	q_2	Remove Blue plate and remain in state q_2	-	-
Green	q_1	Add a Blue plate and remain in state q_1	Add Green plate and remain in state q_1	Go to state q_2
	q_2	-	Remove Green plate and remain in state q_2	-
Red	q_1	Add a Blue plate and remain in state q_1	Add Green plate and remain in state q_1	Go to state q_2
	q_2	Without waiting for input remove Red plate		

- **Example:** 0^n1^n, n>=0

  $M = (\{q_1, q_2\}, \{0, 1\}, \{L, \#\}, \delta, q_1, \#, \varnothing)$

  δ:

  (1) $\delta(q_1, 0, \#) = \{(q_1, L\#)\}$ // stack order: L on top, then # below

  (2) $\delta(q_1, 1, \#) = \varnothing$ // illegal, string rejected, *When will it happen?*

  (3) $\delta(q_1, 0, L) = \{(q_1, LL)\}$

  (4) $\delta(q_1, 1, L) = \{(q_2, \varepsilon)\}$

  (5) $\delta(q_2, 1, L) = \{(q_2, \varepsilon)\}$

  (6) $\delta(q_2, \varepsilon, \#) = \{(q_2, \varepsilon)\}$ //if ε read & stack hits bottom, accept

  (7) $\delta(q_2, \varepsilon, L) = \varnothing$ // illegal, string rejected

  (8) $\delta(q_1, \varepsilon, \#) = \{(q_2, \varepsilon)\}$ // n=0, accept

- **Goal:** (acceptance)
  - Read the entire input string
  - Terminate with an empty stack

- Informally, a string is accepted if there exists a computation that uses up all the input and leaves the stack empty.
- *How many rules should be there in delta?*

- **Language: 0^n1^n, n>=0**

  δ:

  (1) $\delta(q_1, 0, \#) = \{(q_1, L\#)\}$ // stack order: L on top, then # below

  (2) $\delta(q_1, 1, \#) = \emptyset$ // illegal, string rejected, *When will it happen?*

  (3) $\delta(q_1, 0, L) = \{(q_1, LL)\}$

  (4) $\delta(q_1, 1, L) = \{(q_2, \varepsilon)\}$

  (5) $\delta(q_2, 1, L) = \{(q_2, \varepsilon)\}$

  (6) $\delta(q_2, \varepsilon, \#) = \{(q_2, \varepsilon)\}$ //if ε read & stack hits bottom, accept

  (7) $\delta(q_2, \varepsilon, L) = \emptyset$ // illegal, string rejected

  (8) $\delta(q_1, \varepsilon, \#) = \{(q_2, \varepsilon)\}$ // n=0, accept

- **0011**
- ***(q1, 0 011, #) |-***

  (q1, 0 11, L#) |-

  (q1, 1 1, LL#) |-

  (q2, 1, L#) |-

  (q2, e, #) |-

  (q2, e, e): accept

- ***011***
- ***(q1, 0 11, #) |-***

  (q1, 1 1, L#) |-

  (q2, 1, #) |-

  Ø : reject

- ***Try 001***

- **Example:** balanced parentheses,
- e.g. in-language: ((())()), or (())(), but not-in-language: ((())

$M = (\{q_1\}, \{\text{“(“}, \text{“)”}\}, \{L, \#\}, \delta, q_1, \#, \varnothing)$

δ:

(1) $\delta(q_1, (, \#) = \{(q_1, L\#)\}$ // stack order: L-on top-then- # lower

(2) $\delta(q_1,), \#) = \varnothing$ // illegal, string rejected

(3) $\delta(q_1, (, L) = \{(q_1, LL)\}$

(4) $\delta(q_1,), L) = \{(q_1, \varepsilon)\}$

(5) $\delta(q_1, \varepsilon, \#) = \{(q_1, \varepsilon)\}$ //if ε read & stack hits bottom, accept

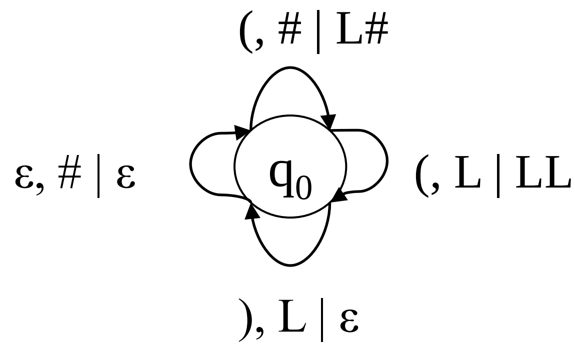
(6) $\delta(q_1, \varepsilon, L) = \varnothing$ // illegal, string rejected

// *What does it mean? When will it happen?*

- **Goal:** (acceptance)
  - Read the entire input string
  - Terminate with an empty stack

- Informally, a string is accepted if there exists a computation that uses up all the input and leaves the stack empty.
- *How many rules should be in delta?*

- Transition Diagram:



- Example Computation:

Current Input	Stack	Transition	
(())	#	-- initial status	
())	L#	(1)	- Could have applied rule (5), but
))	LL#	(3)	it would have done no good
)	L#	(4)	
ε	#	(4)	
ε	-	(5)	

Thank You.