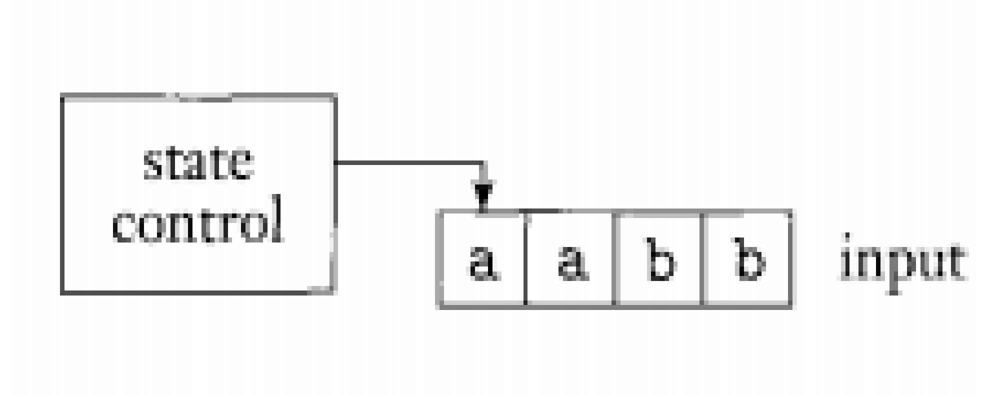
PUSHDOWN AUTOMATA

INTRODUCTION

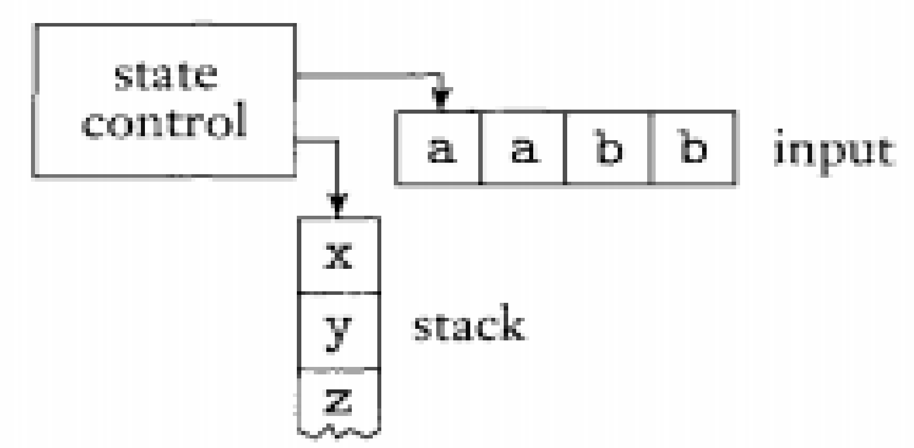
- Pushdown automata is another type of a computational model
- These automata are like nondeterministic finite automata but have an extra component called a stack.
- The stack provides additional memory beyond the finite amount available in the control.
- The stack allows pushdown automata to recognize some non-regular languages.
- A stack is valuable since it can hold an unlimited amount of information .
- This makes the PDA store a lot of information thus being able to recognize the language $\{0^n1^n \mid n \geq 0\}$

PUSHDOWN AUTOMATA

- In a figure a) , the **control** represents the states and transition function, the tape contains the input string, and the arrow represents the input head, pointing at the next input symbol to be read
- With the addition of a stack component we obtain a schematic representation of a pushdown automaton, as shown in the following figure.



a) Finite Automaton



b) Pushdown Automaton

FORMAL DEFINITION OF A PUSHDOWN AUTOMATON

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

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta \colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

FORMAL DEFINITION OF A PUSHDOWN AUTOMATON

A pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ computes as follows. It accepts input w if w can be written as $w = w_1 w_2 \cdots w_m$, where each $w_i \in \Sigma_\varepsilon$ and sequences of states $r_0, r_1, \ldots, r_m \in Q$ and strings $s_0, s_1, \ldots, s_m \in \Gamma^*$ exist that satisfy the following three conditions. The strings s_i represent the sequence of stack contents that M has on the accepting branch of the computation.

1. $r_0 = q_0$ and $s_0 = \varepsilon$. This condition signifies that M starts out properly, in the start state and with an empty stack.
2. For $i = 0, \ldots, m-1$, 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 $t \in \Gamma^*$. This condition states that M moves properly according to the state, stack, and next input symbol.
3. $r_m \in F$. This condition states that an accept state occurs at the input end.

Example 1

- The following is the formal description of a PDA that recognizes the language $\{0^n1^n \mid n \geq 0\}$. Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where

$Q = \{q_1, q_2, q_3, q_4\}$,

$\Sigma = \{0,1\}$,

$\Gamma = \{0, \$\}$,

$F = \{q_1, q_4\}$, and

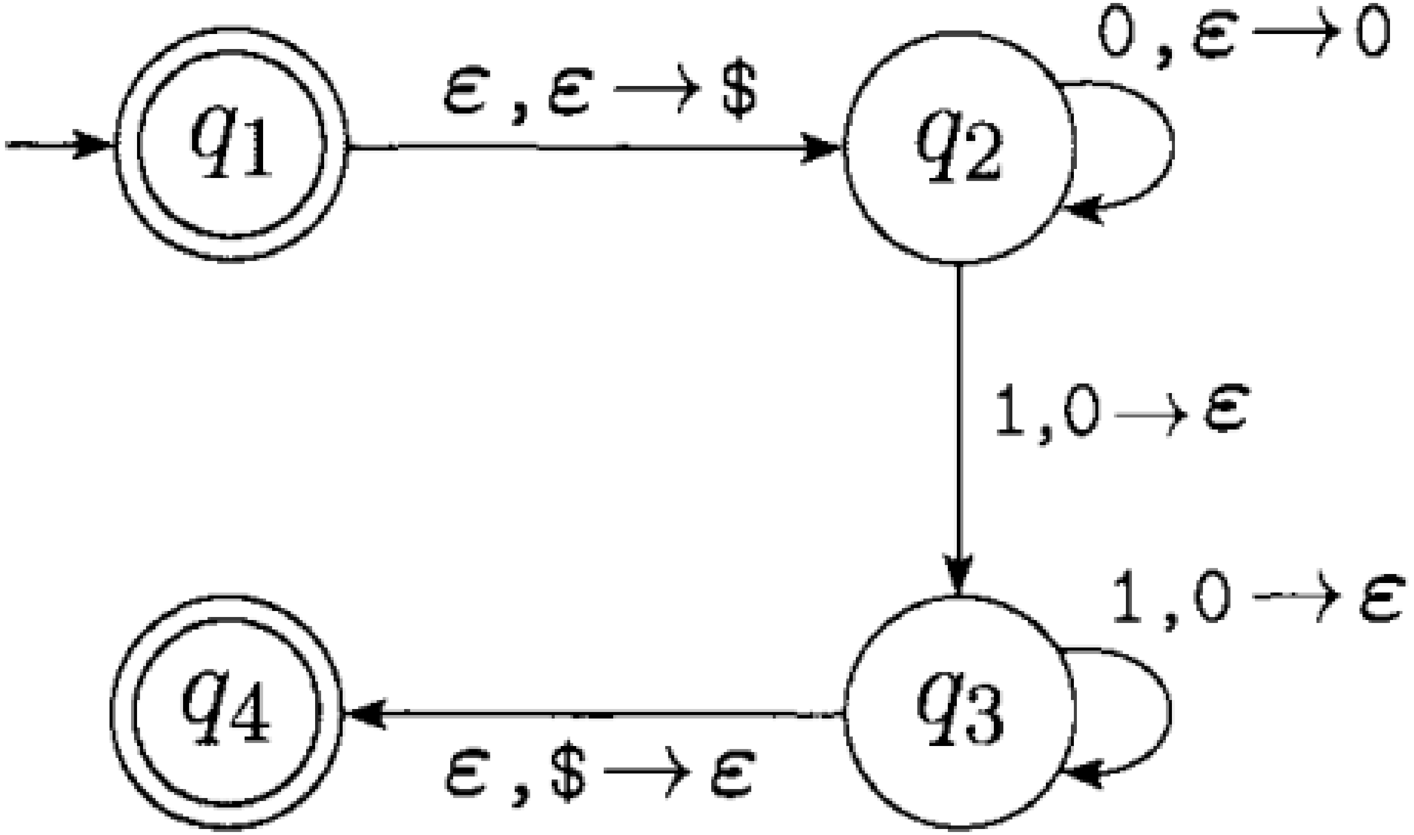
δ is given by the following table, wherein blank entries signify $\emptyset$.

Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

Example 1

- We write "***a,b* → *c***" to signify that when the machine is reading an a from the input it may replace the symbol ***b*** on the top of the stack with a ***c***.
- Any of ***a***, ***b***, and ***c*** may be ε.
  - ❑ If ***a*** is ε, the machine may make this transition without reading any symbol from the input.
  - ❑ If ***b*** is ε, the machine may make this transition without reading and popping any symbol from the stack.
  - ❑ If ***c*** is ε, the machine does not write any symbol on the stack when going along this transition.

Example 1



State diagram for the PDA M_1 that recognizes $\{0^n 1^n \mid n \geq 0\}$

IN SUMMARY

- All regular languages are context-free languages