April 3 Test2
April 29 Test3

GREIBACH NORMAL FORM (GNF)

COMP 4200 – Formal Language

GNF not on exam

GREIBACH NORMAL FORM (GNF)

- Every context free grammar/ CFL without ε can be generated by a grammar for which every production in the form of $A \rightarrow a\alpha$ and $A \rightarrow a$ where A is variable, a is a terminal and α is collection of variables
- GNF FORMAT
  - Non-terminal → terminal .any no of non-terminal
  - Non-terminal → terminal
- A CFG is in GNF if the productions are in form,
  - $A \rightarrow b$
  - $A \rightarrow bC_1C_2.....C_n$

$A \rightarrow aA, A_2 \cdots$

$A \rightarrow b$

- Steps:
  - Check if the given CFG has any UNIT productions or NULL productions and remove if any.
  - Check if CFG is already in CNF and convert it to CNF if not.
  - Change the names of nonterminal symbols into some A_i in ascending order of i.
- Example:
  - S→CA/BB
  - B→b/SB
  - C→b
  - A→a
- Step 1 and 2 are satisfied.
- Step 3: S,B,C,A → non-terminal symbols.
  - Replace, S with A_1
  - C with A_2
  - A with A_3
  - B with A_4

$A_1 \rightarrow A_2A_3/A_4A_4$
$A_4 \rightarrow b/A_1A_4$
$A_2 \rightarrow b$
$A_3 \rightarrow a$

- Step 4: After the rules so that the non-terminal are in ascending order such that, if the production is of the form $A_i \rightarrow A_jX$ then, i<j and should never be i>j.
  - $A_1 \rightarrow A_2A_3/A_4A_4$ => (1<2 and 1<4) => ☑
  - $A_4 \rightarrow b/A_1A_4$ => (4>1) => we need to resolve this.
  - Replace A_1
  - $A_4 \rightarrow b/ A_2A_3A_4/A_4A_4\,A_4$ => (4>2) => again we need to resolve this issue.
  - Replace value of A_2
  - $A_4 \rightarrow b/ bA_3A_4/A_4A_4\,A_4$ => (4≥4) => this needs to be resolved, called Left recursion.

    This is GNF

- Step 5: Remove Left Recursion
  - Introduce new variable to remove left recursion
  - $A_4 \rightarrow b/ bA_3A_4/A_4A_4\,A_4$
  - $Z \rightarrow A_4A_4Z/ A_4A_4$ => take variable that follow problematic ones and write it along with new variable.
  - Rewriting,
  - $A_4 \rightarrow b/ bA_3A_4/bZ/bA_3\,A_4Z$

- Step 5 continuation: We are not allowed to have variable in beginning
  - $A_1 \rightarrow A_2A_3/A_4\,A_4$
  - $A_4 \rightarrow b/\ bA_3A_4/bZ/bA_3\,A_4Z$
  - $Z \rightarrow A_4A_4Z/\ A_4A_4$
  - $A_2 \rightarrow b$
  - $A_3 \rightarrow a$
- Modifying:
  - $A_1 \rightarrow A_2A_3/A_4\,A_4$ => replace A_4
  - $A_1 \rightarrow bA_3/\ bA_4/bA_3\,A_4A_4/bZA_4/bA_3\,A_4ZA_4$
  - $A_4 \rightarrow b/\ bA_3A_4/bZ/bA_3\,A_4Z$
  - $Z \rightarrow bA_4/\ bA_3\,A_4A_4/bZA_4/bA_3\,A_4ZA_4/bA_4A_4z/bA_3A_4A_4Z/bzA_4Z/bA_3A_4ZA_4Z$
  - $A_2 \rightarrow b$
  - $A_3 \rightarrow a$

TRY YOURSELF!

- Convert the given CFG to GNF:
  - S→CA
  - A→a
  - C→aB/b
- CFG to GNF
  - S → AA | 0
  - A → SS | 1
- Convert the given CFG to GNF:
  - S→AB
  - A→BS/b
  - B→ SA

- CFG to GNF
  - S → AA | 0
  - A → SS | 1

$S \to A_1$

$A \to A_2$

$A_1 \to A_2 A_2 \mid 0$

$A_2 \to A_1 A_1 \mid 1$

$A_i := A_j X$

$A_1 \to A_2 A_2 \mid 0$ ✓ $[1<2]$

$A_2 \to A_1 A_1 \mid 1$ $[2>1]$

$A_2 \to [A_2 A_2 \mid 0] A_1 \mid 1 \to A_2 A_2 A_1 \mid 0 A_1 \mid 1$

Left Recursion

$Z \to A_2 A_1 \mid A_2 A_1 Z$

$A_2 \to 0A_1 \mid 1 \mid 0A_1 Z \mid 1Z$

$A_1 \to A_2 A_2 \mid 0$

$\to [0A_1 \mid 1 \mid 0A_1 Z \mid 1Z] A_2 \mid 0$

$\to 0A_1 A_2 \mid 1A_2 \mid 0A_1 Z A_2 \mid 1Z A_2 \mid 0$

$Z \rightarrow 0A_1 \mid 1 \mid 0A_1Z \mid 1Z \mid A_1 \mid [0A_1 \mid 1 \mid 0A_1Z \mid 1Z]A_1Z$

$Z \rightarrow 0A_1A_1 \mid 1A_1 \mid 0A_1ZA_1 \mid 1ZA_1 \mid 0A_1A_1Z \mid 1A_1Z \mid 0A_1ZA_1Z \mid 1ZA_1Z$

- Convert the given CFG to GNF:
  - S→AB
  - A→BS/b
  - B→ SA

$S \rightarrow A_1$

$A \rightarrow A_2$

$B \rightarrow A_3$

$A_i = A_j X$

$i < j$

$A_1 \rightarrow A_2A_3$ ✓

$A_2 \rightarrow A_3A_1 \mid b$ ✓

$A_3 \rightarrow A_1A_2 \rightarrow A_3 \rightarrow [A_2A_3]A_2 \rightarrow A_2A_3A_2$

$\rightarrow [A_3A_1 \mid b]A_3A_2 \rightarrow \underbrace{A_3A_1A_3A_2}_{\text{left recursion}} \mid bA_3A_2$

$Z \rightarrow A_1A_3A_2 \mid A_1A_3A_2Z$

$A_3 \rightarrow bA_3A_2 \mid bA_3A_2Z$

$A_1 \rightarrow A_2A_3$

$A_2 \rightarrow A_3A_1 \mid b$

$A_2 \rightarrow bA_3A_2A_1 \mid bA_3A_2ZA_1 \mid b$

$A_1 \rightarrow bA_3A_2A_1A_3 \mid bA_3A_2ZA_1A_3$

$Z \rightarrow bA_3A_2A_1A_3A_3A_2$

EXAMPLE

- S → AB A → BSB A → BB A → a Ab B → a A → b B → a A → b

$S \to AB$

$A \to XB \mid BB \mid a \mid b$

$B \to a$

$X \to BS$

$S \to A_1$

$A \to A_2$

$B \to A_3$

$X \to A_4$

$A_1 \to A_2 A_3$ ✓

$A_2 \to A_4 A_3 \mid A_3 A_3 \mid a \mid b$ ✓ ✓

$i < j$

$A_3 \to a$ ✓

$A_4 \rightarrow A_3 A_1$ X $\rightarrow a A_1$

$A_4 \rightarrow a A_1$

$A_3 \rightarrow a$

$A_2 \rightarrow aA_1A_3 \mid aA_3 \mid a \mid b$

$A_1 \rightarrow aA_1A_3A_3 \mid aA_3A_3 \mid aA_3 \mid bA_3$

PUSHDOWN AUTOMATA

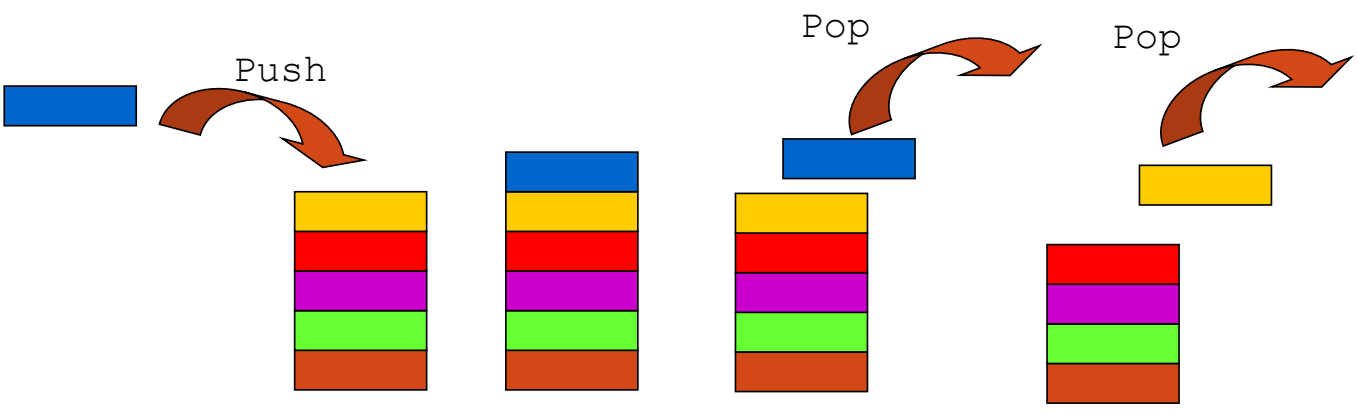
PUSHDOWN AUTOMATA(PDA)

- Pushdown automata (PDAs) are for CFLs what finite automata are for regular languages.
- PDAs are like nondeterministic finite automata but have an extra component called a stack.
- PDA is presented with a string w over an alphabet Σ. PDA accepts or doesn't accept w.
- More powerful than FA.
- Key Differences Between PDA and DFA:
  - PDAs have a single stack.
  - PDAs allow for nondeterminism

STACK REVIEW

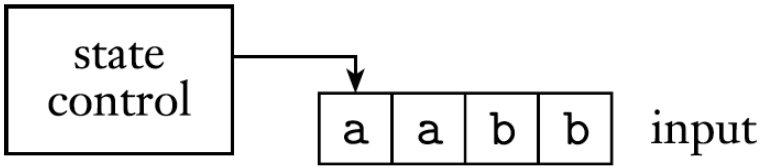
- Defn: Stack is data structure of unlimited size with 2 operations.
- Operations:
  - Push → adds item to top of stack,
  - Pop → removes item from top of stack.
- Last-In-First-Out (LIFO)

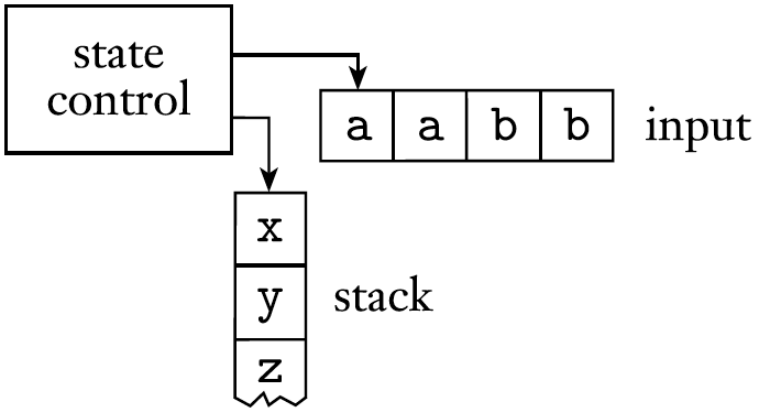
STACK



- Writing a symbol on the stack is referred to as pushing the symbol
- Removing a symbol from the stack is referred to as popping the symbol
- All access to the stack may be done only at the top

FA vs PDA





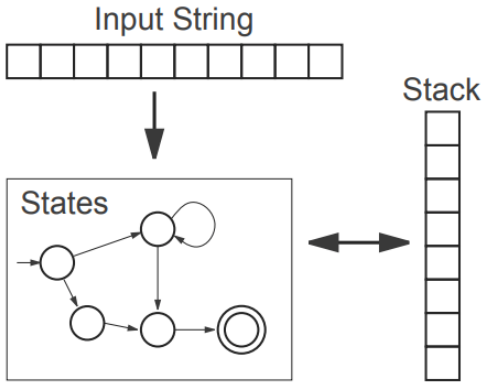
PDA is a way to implement a CFG.

1. More powerful than FA
2. FA has very limited memory while PDA has more storage
3. PDA = FA + Stack
    1. FA → mostly NFA
    2. Stack → Allows PDA to recognize some non regular language

Deterministic and nondeterministic pushdown automata are not equivalent in power.

PDA

- PDA has
  - States
  - Stack with alphabet Γ
  - Transitions among states based on
    - current state
    - what is read from input string
    - what is popped from stack.
  - At end of each transition, symbol may be pushed on stack.



INFORMAL DESCRIPTION

- $\{0^n 1^n \mid n \geq 0\}$
- Read symbols from the input.
- As each 0 is read, push it onto the stack.
- As soon as 1s are seen, pop a 0 off the stack for each 1 read.
- If reading the input is finished exactly when the stack becomes empty of 0s, accept the input.
- If the stack becomes empty while 1s remain or if the 1s are finished while the stack still contains 0s or if any 0s appear in the input following 1s, reject the input.

USE OF STACK

- Recall for alphabet Σ, we defined $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.
- Let Γ be stack alphabet
  - Symbols in Γ can be pushed onto and popped off stack.
  - Often have $\$ \in \Gamma$ to mark bottom of stack.
- Let $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$.
  - Pushing or popping ε leaves stack unchanged.

DEFINITION OF PDA

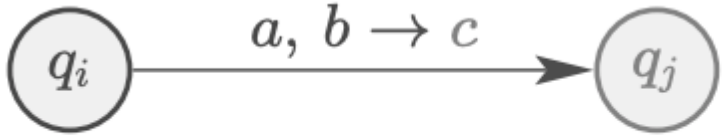
- Defn: Pushdown automaton (PDA) is a 6 tuple ,

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

- Q is finite set of states
- Σ is (finite) input alphabet
- Γ is (finite) stack alphabet → always includes ε
- q_0 is start state, $q_0 \in Q$
- F is set of accept states, $F \subseteq Q$
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$ is transition function

PDA TRANSITIONS

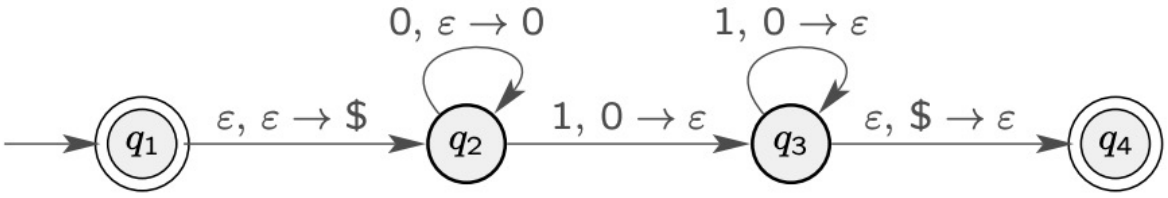
- If PDA
  - currently in state q_i,
  - reads a $\in \Sigma_\varepsilon$, and
  - pops b $\in \Gamma_\varepsilon$ off the stack,
- then PDA can
  - move to state q_j
  - push c $\in \Gamma_\varepsilon$ onto top of stack
- If a = ε, then no input symbol is read.
- If b = ε, then nothing is popped off stack.
- If c = ε, then nothing is pushed onto stack.



read, pop → push

$ is end of stack read, pop → push

HOW A PDA COMPUTES?



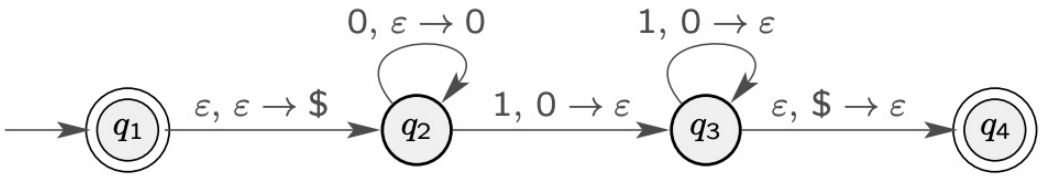
- PDA starts in start state with input string w ∈ Σ*
  - stack initially empty
- PDA makes transitions among states
  - Based on current state, what from Σ_ε is next read from w, and what from Γ_ε is popped from stack.
  - Nondeterministically move to state and push from Γ_ε onto stack.
- If possible, to end in accept state ∈ F ⊆ Q after reading entire input w without crashing, then M accepts w.

INSTANTANEOUS DESCRIPTION(ID)

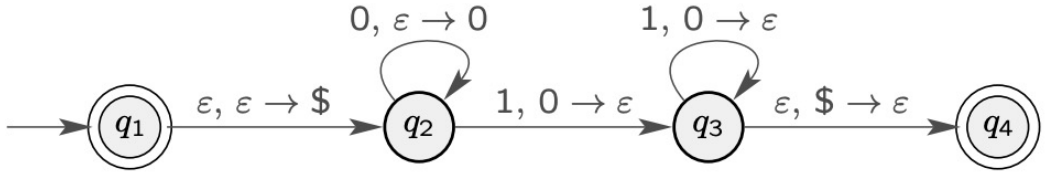
- It describes the configuration of the PDA at a given instance.
- The ID must record the state and stack contents.
- An ID is a format of triple (q, w, κ), where
  - q ∈ Q (finite set of states),
  - w ∈ Σ (finite set of input alphabets), and
  - κ ∈ Γ (finite set of stack symbols).

EXAMPLE

- PDA M = (Q, Σ, Γ, δ, q1, F)



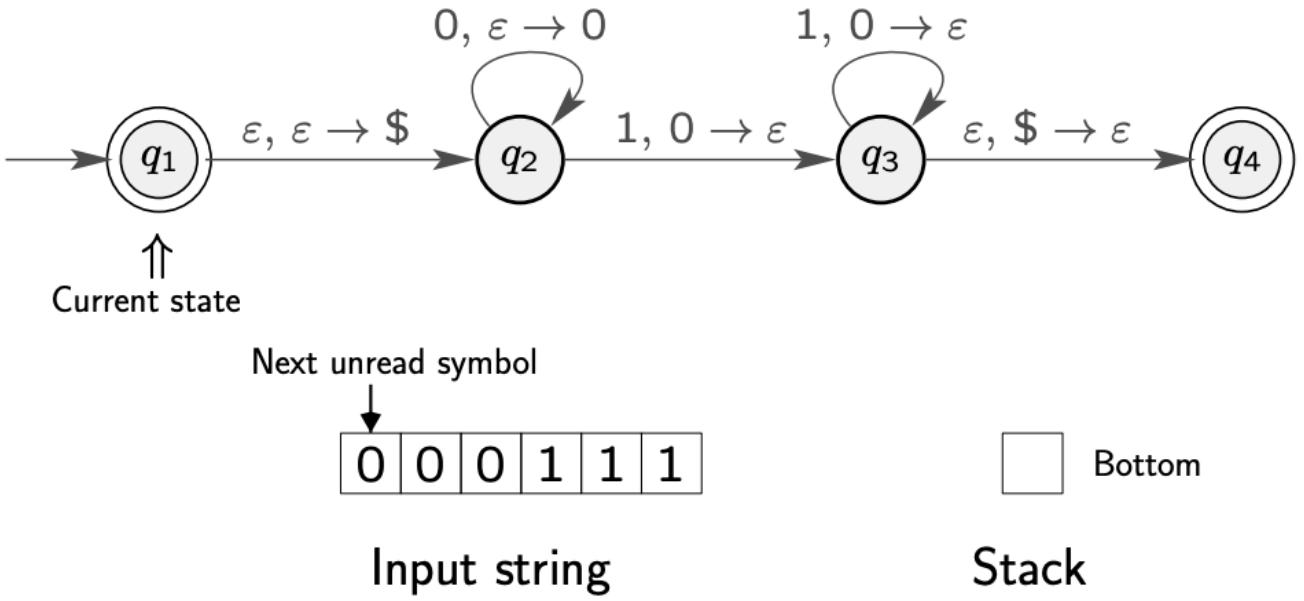
- Q = {q1, q2, q3, q4}
- Σ = {0, 1}
- Γ = {0, $} (use $ to mark bottom of stack)
- q1 is the start state
- F = {q1, q4}
- M recognizes language { 0^n1^n | n ≥ 0 }.



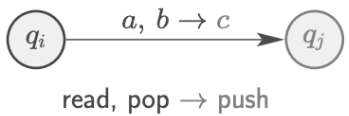
- transition function $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$

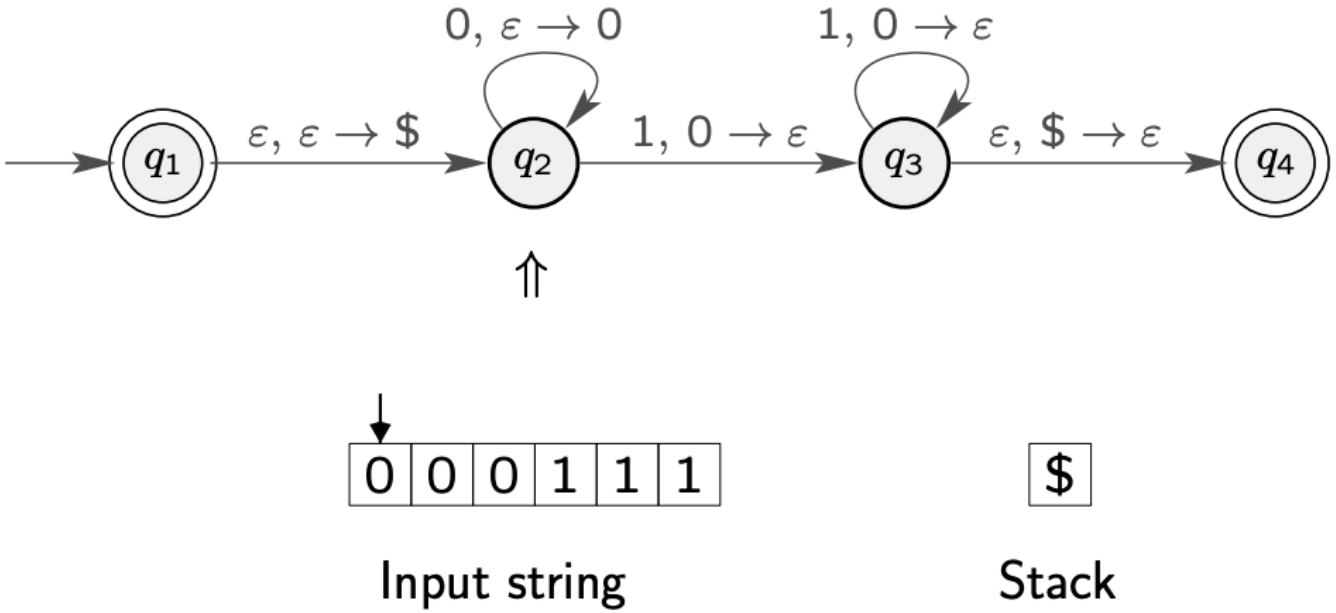
Input:	0			1			ε		
Stack:	0	$	ε	0	$	ε	0	$	ε
q_1									$\{ (q_2, \$) \}$
q_2			$\{ (q_2, 0) \}$	$\{ (q_3, \varepsilon) \}$					
q_3				$\{ (q_3, \varepsilon) \}$				$\{ (q_4, \varepsilon) \}$	
q_4									

  - e.g., $\delta(q_2, 1, 0) = \{ (q_3, \varepsilon) \}$.
  - Blank entries are $\emptyset$.
- Let's process string 000111 on our PDA.
- PDA uses stack to match each 0 to a 1

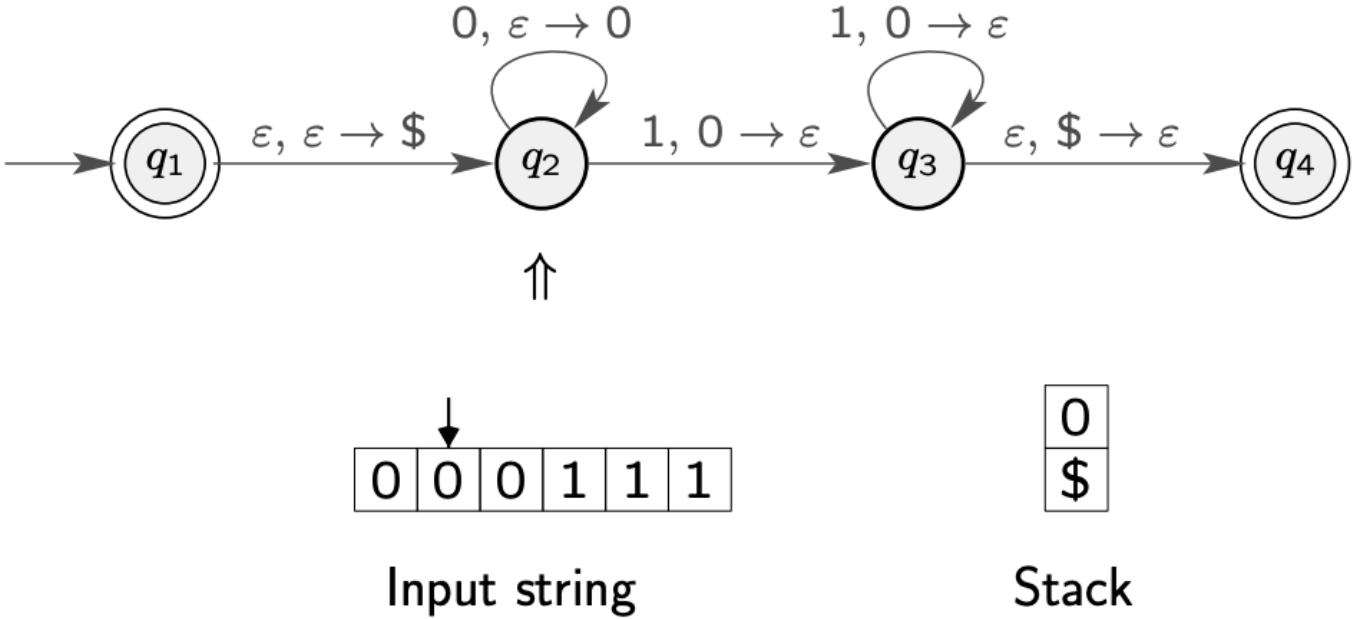


- Start in start state q_1 with stack empty.
- No input symbols read so far.
- Next go to state q_2
  - reading nothing, popping nothing, and pushing $ on stack.

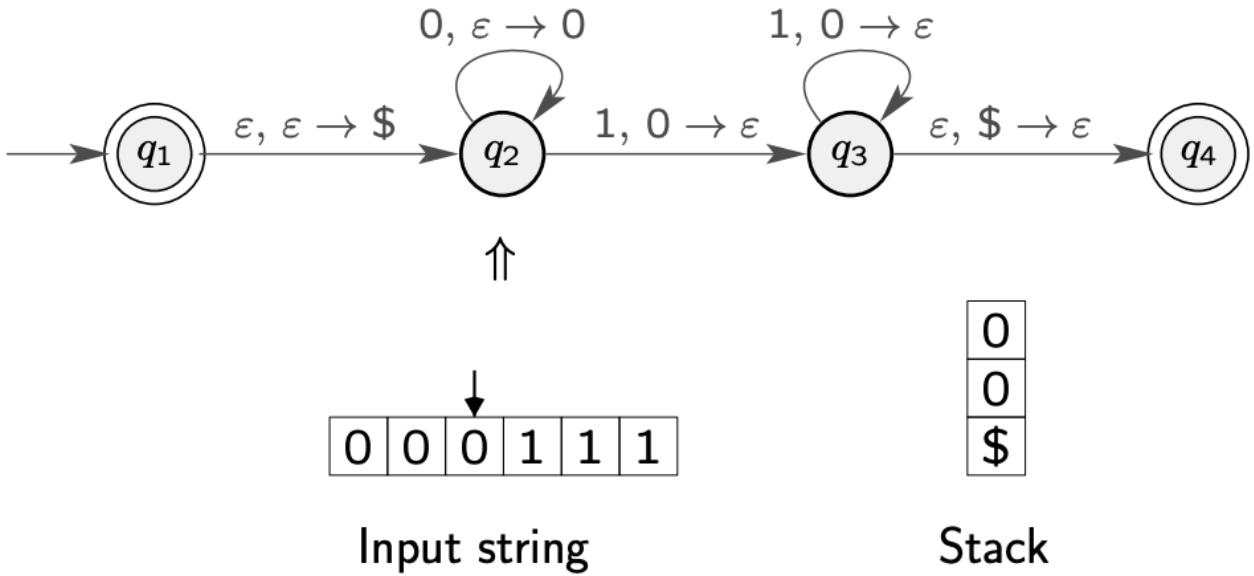




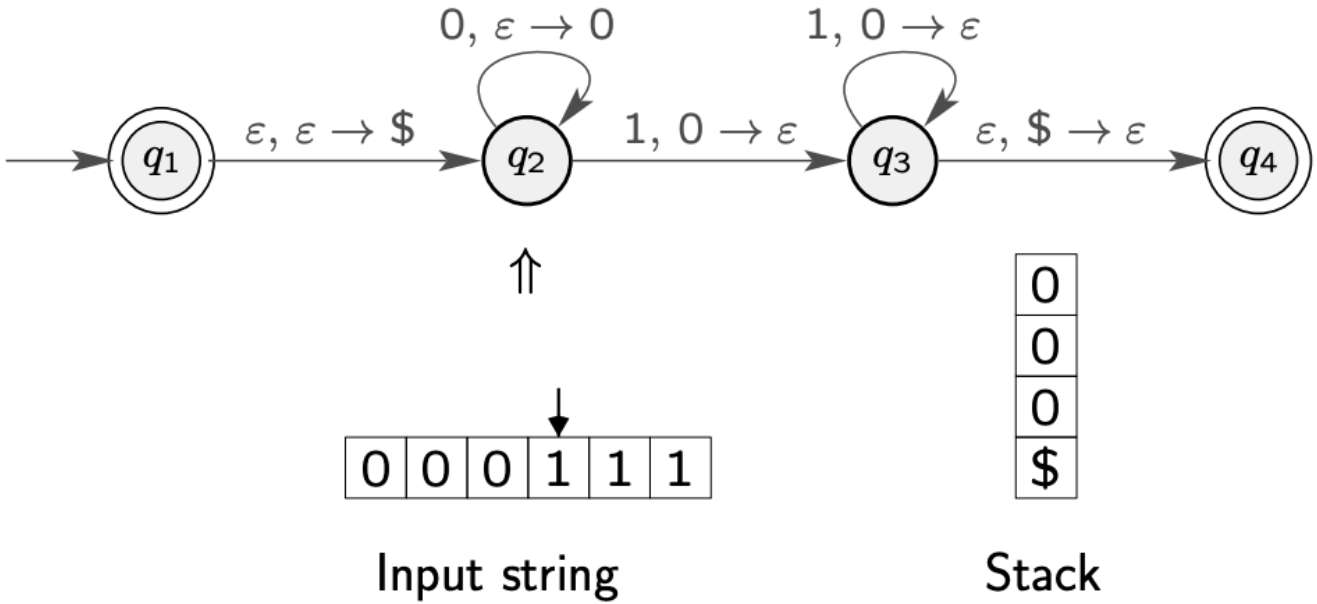
- Next return to state q_2
  - reading input symbol 0
  - popping nothing from stack
  - pushing 0 on stack.



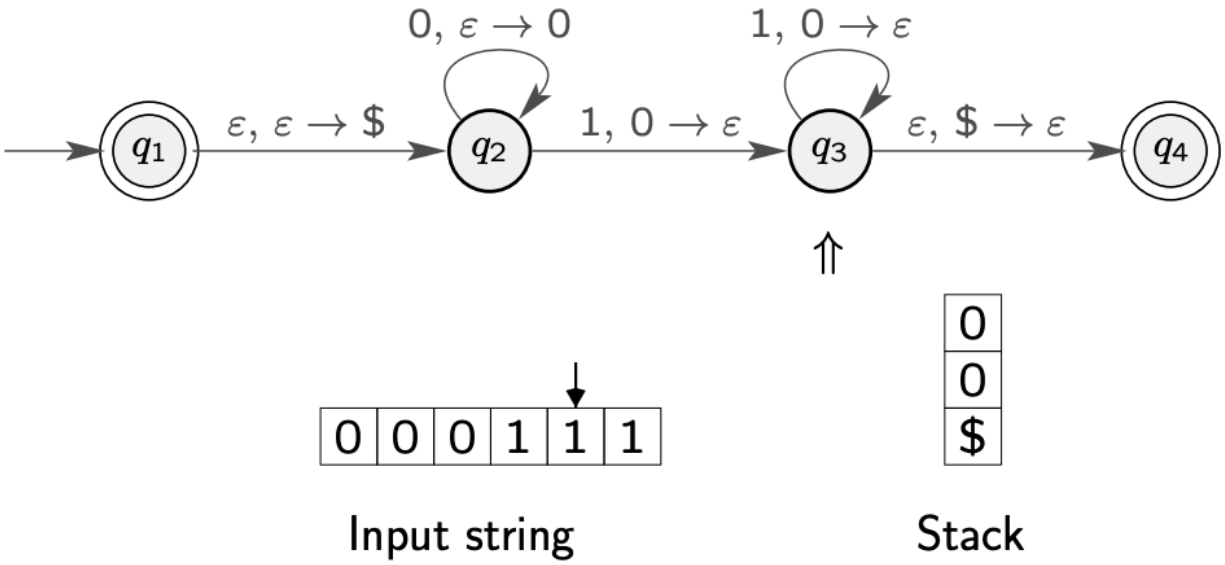
- Next return to state q_2
  - reading input symbol 0
  - popping nothing from stack
  - pushing 0 on stack.



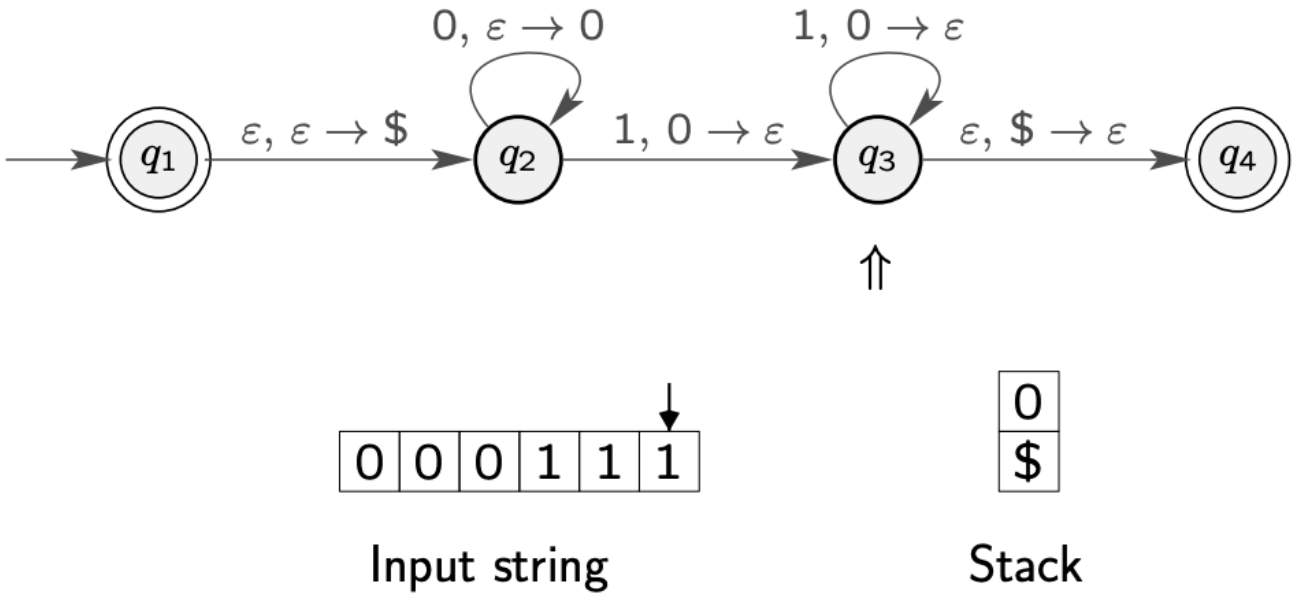
- Next return to state q_2
  - reading input symbol 0
  - popping nothing from stack
  - pushing 0 on stack.



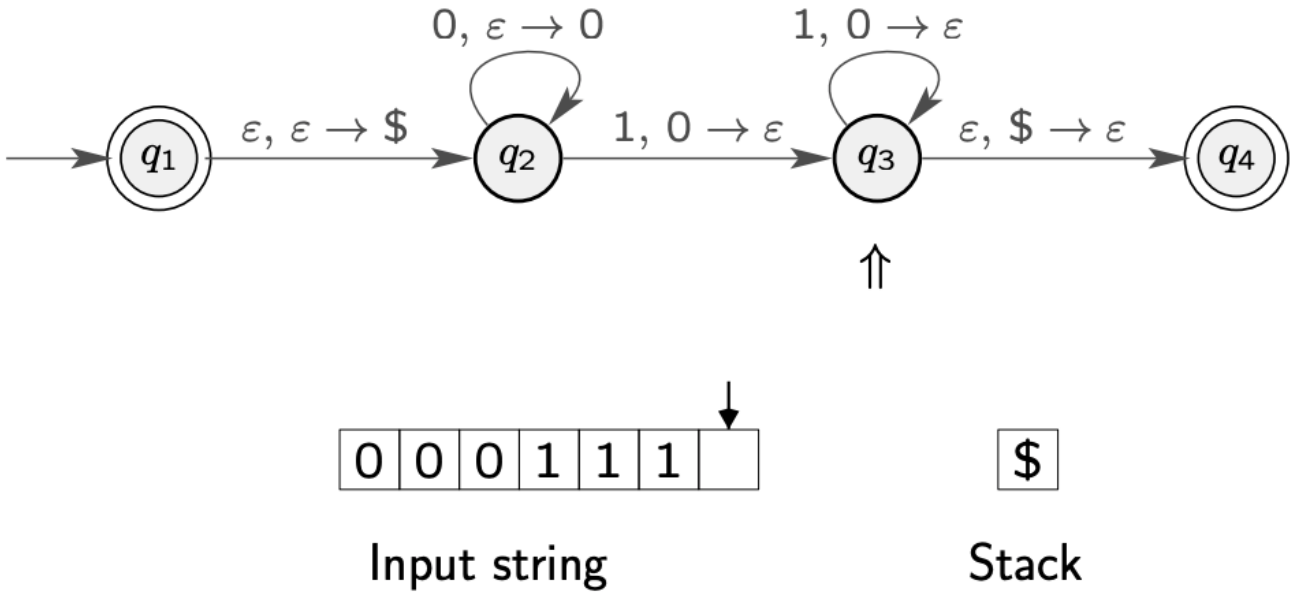
- Next go to state q_3
  - reading input symbol 1
  - popping 0 from stack
  - pushing nothing on stack.



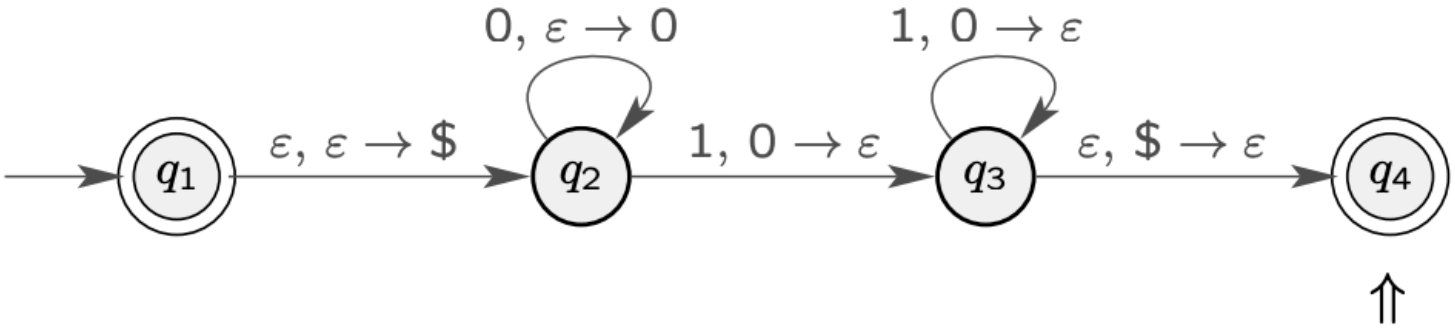
- Next return to state q_3
  - reading input symbol 1
  - popping 0 from stack
  - pushing nothing on stack.

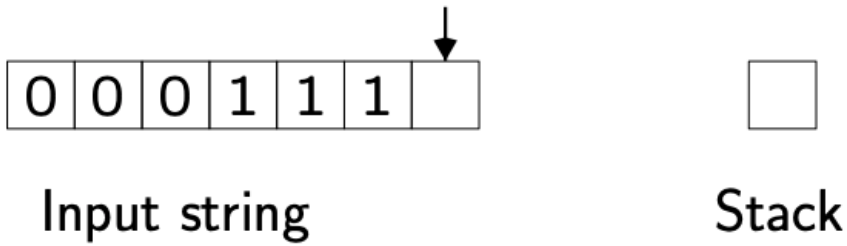


- Next return to state q_3
  - reading input symbol 1
  - popping 0 from stack
  - pushing nothing on stack.

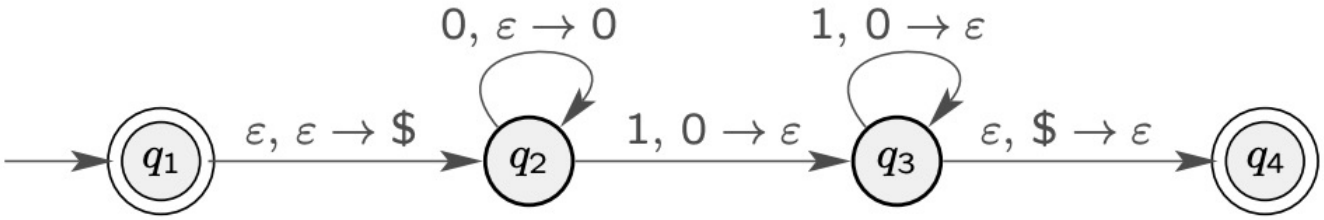


- Next go to state q_4
  - reading nothing
  - popping $ from stack
  - pushing nothing on stack.





- String 000111 is accepted by PDA because
  - q_4 is an accept state and
  - PDA read the entire input string without crashing.



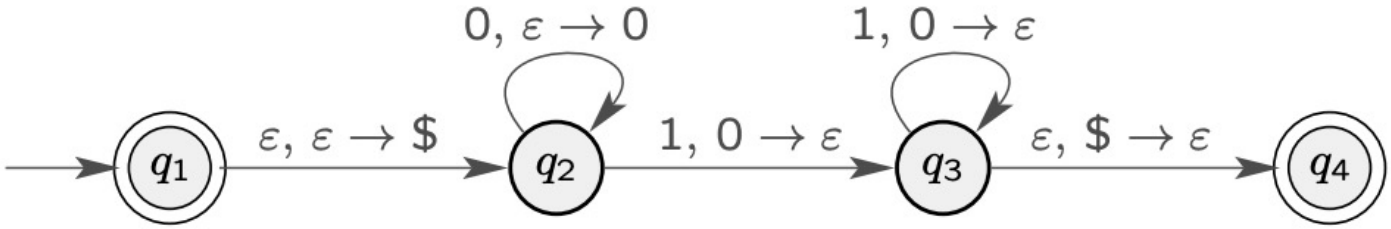
On input $w = 000111$, the (state; stack) evolution is

$$(q_1;\varepsilon) \xrightarrow{\varepsilon,\varepsilon\to\$} (q_2;\$) \xrightarrow{0,\varepsilon\to 0} (q_2;0\$) \xrightarrow{0,\varepsilon\to 0} (q_2;00\$)$$

$$\xrightarrow{0,\varepsilon\to 0} (q_2;000\$) \xrightarrow{1,0\to\varepsilon} (q_3;00\$) \xrightarrow{1,0\to\varepsilon} (q_3;0\$) \xrightarrow{1,0\to\varepsilon} (q_3;\$)$$

$$\xrightarrow{\varepsilon,\$\to\varepsilon} (q_4;\varepsilon).$$

- Stack grows to the left, so leftmost symbol in stack is on top.
- Concatenation of what is read in sequence of transitions is $\varepsilon 000111\varepsilon$ = w.



- On input $w = 0111$, the (state; stack) evolution is

$$(q_1; \varepsilon) \xrightarrow{\varepsilon,\varepsilon\to\$} (q_2; \$) \xrightarrow{0,\varepsilon\to 0} (q_2; 0\$) \xrightarrow{1,0\to\varepsilon} (q_3; \$) \xrightarrow{\varepsilon,\$\to\varepsilon} (q_4; \varepsilon)$$

- Only first two symbols 01 were read from input w = 0111.
- PDA then crashes: there are still unread symbols 11 in input string w

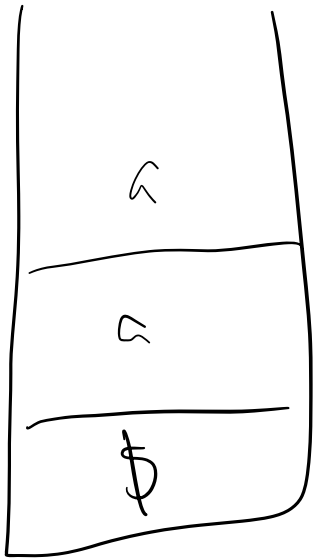
but PDA can't make any more transitions from q_4.

- No other way of processing, so string 0111 not accepted.
- Can show that PDA M recognizes language $\{ 0^n 1^n \mid n \geq 0 \}$.

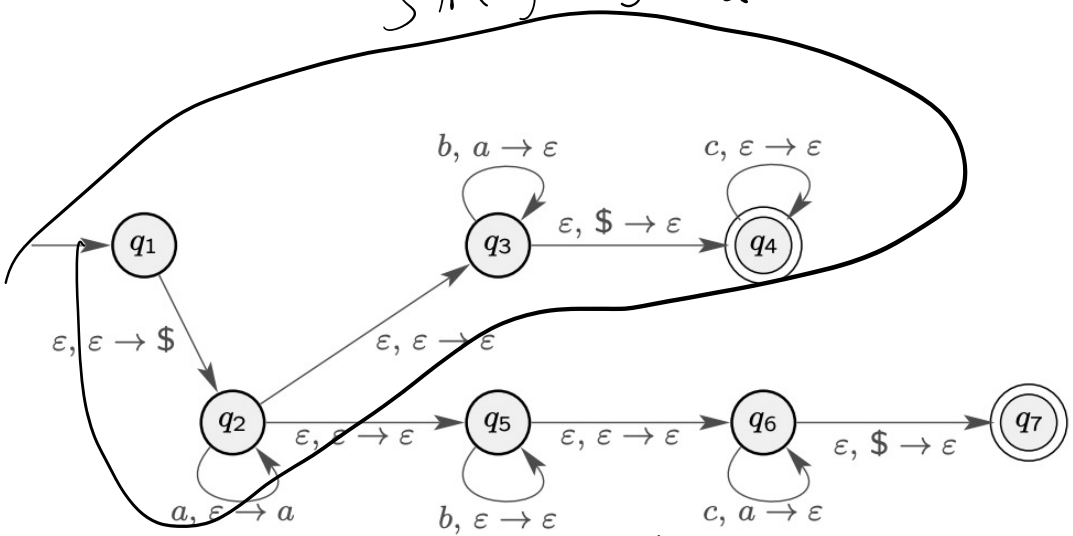
show stack representation

a a b b $

EXAMPLE



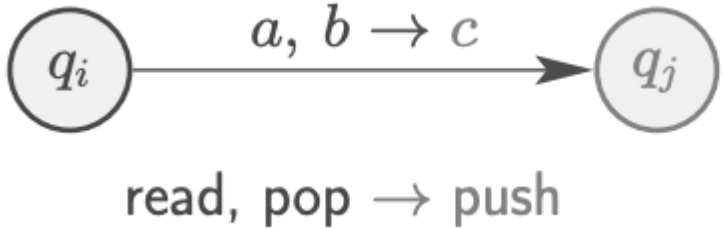
string rejected



double rejected by machine

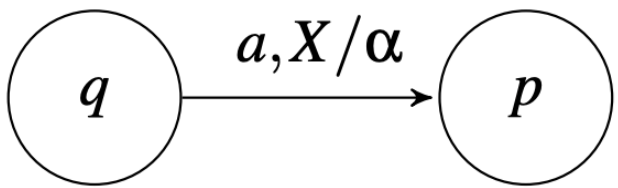
PDA TRANSITIONS

- If PDA
  - currently in state q_i,
  - reads a $\in \Sigma_\varepsilon$, and
  - pops b $\in \Gamma_\varepsilon$ off the stack,
- then PDA can
  - move to state q_j
  - push c $\in \Gamma_\varepsilon$ onto top of stack
- If a = ε, then no input symbol is read.
- If b = ε, then nothing is popped off stack.
- If c = ε, then nothing is pushed onto stack.



ANOTHER REPRESENTATION

- The nodes correspond to the states of the PDA.
- An arrow labelled start indicates the start state.
- Doubly circled states are accepting.
- If $\delta(q,a,X)$ contains a pair (p,α), then there is an arc from q to p labeled $a,X/\alpha$



- The only thing that the diagram does not tell us is which stack symbol is the start symbol. Conventionally, it is Z.

EXAMPLE

The diagramm to our PDA for $\{a^n b^n \mid n \geq 1\}$

EXAMPLE

The diagramm to our PDA for $\{a^n b^n \mid n \geq 1\}$

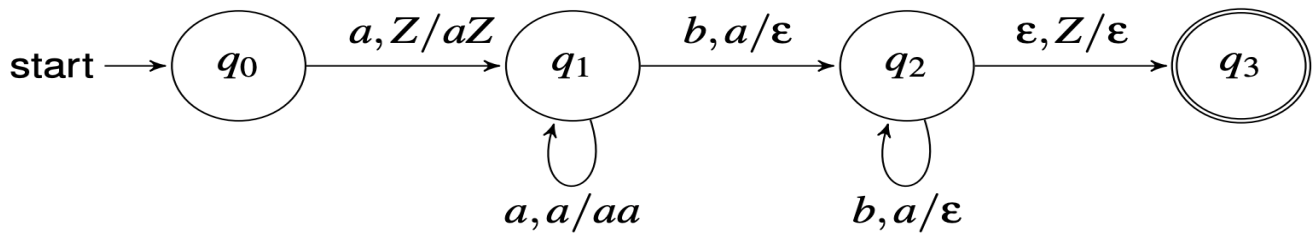
$\delta(q_0, a, Z) = \{(q_1, aZ)\}$ ··· push first a

$\delta(q_1, a, a) = \{(q_1, aa)\}$ ··· push further a's

$\delta(q_1, b, a) = \{(q_2, \varepsilon)\}$ ··· start popping a's

$\delta(q_2, b, a) = \{(q_2, \varepsilon)\}$ ··· pop further a's

$\delta(q_2, \varepsilon, Z) = \{(q_3, \varepsilon)\}$ ··· accept



EXAMPLE

- Transition Function:

1. $q0$ (initial state): Read a, push A onto the stack, move to state $q1$
2. $q1$: Read a, push A onto the stack, remain in state $q1$
3. $q1$: Read b, move to state $q2$ without pushing or popping anything from the stack
4. $q2$: Read b, move to state $q3$ without pushing or popping anything from the stack
5. $q3$: Read b, pop A from the stack, move back to state $q1$
6. $q3$: Read a, remain in state $q3$ if there are no more input symbols left
7. $q3$: If ϵ (empty stack) and no input symbols left, move to state $q4$
8. $q4$: If ϵ (empty stack) and no input symbols left, move to state $q5$ (final accepting state)

TRY YOURSELF!

- Build a PDA for the language L = $\{a^n b^{2n} \mid n \geq 1\}$
- Construct a PDA to accept L = (a, b)* with equal number of 'a' and 'b', i.e., $n_a(L) = n_b(L)$
- Build a PDA for the language L = $\{0^n 1^m 0^n \mid m,n \geq 1\}$
- Construct the PDA for the language L=$\{a^n b^m a^n / n,m >= 1\}$
- Construct a PDA to accept the language L = $\{a^n b^m c^m d^n$, where m, n ≥ 1$\}$.

EXAMPLE

Build a PDA for the language $L = \{a^n b^{2n} \mid n \geq 1\}$

EXAMPLE

Build a PDA for the language $L = \{a^n b^{2n} \mid n \geq 1\}$

- Transition Function:

1. $q0$ (initial state): Read a, push A onto the stack, move to state $q1$
2. $q1$: Read a, push A onto the stack, remain in state $q1$
3. $q1$: Read b, move to state $q2$ without pushing or popping anything from the stack
4. $q2$: Read b, move to state $q3$ without pushing or popping anything from the stack
5. $q3$: Read b, pop A from the stack, move back to state $q1$
6. $q3$: Read a, remain in state $q3$ if there are no more input symbols left
7. $q3$: If ϵ (empty stack) and no input symbols left, move to state $q4$(final accepting state)

TRY YOURSELF!

Build a PDA for the language $L = \{0^n\ 1^m\ 2^n \mid m,n \geq 1\}$