PUSHDOWN AUTOMATA

INTRODUCTION TO PDAs

- A context-free grammar can generate or derive the strings of the context-free language it represents by using the different rules of the grammar.

- In the same manner, a regular expression can be used to generate the strings of the regular language it represents.

- Therefore, regular expressions and context-free grammars are called ***specification mechanisms*** for regular languages and context-free languages, respectively.

INTRODUCTION TO PDAs

- A finite automaton is a mathematical model that recognizes regular languages.

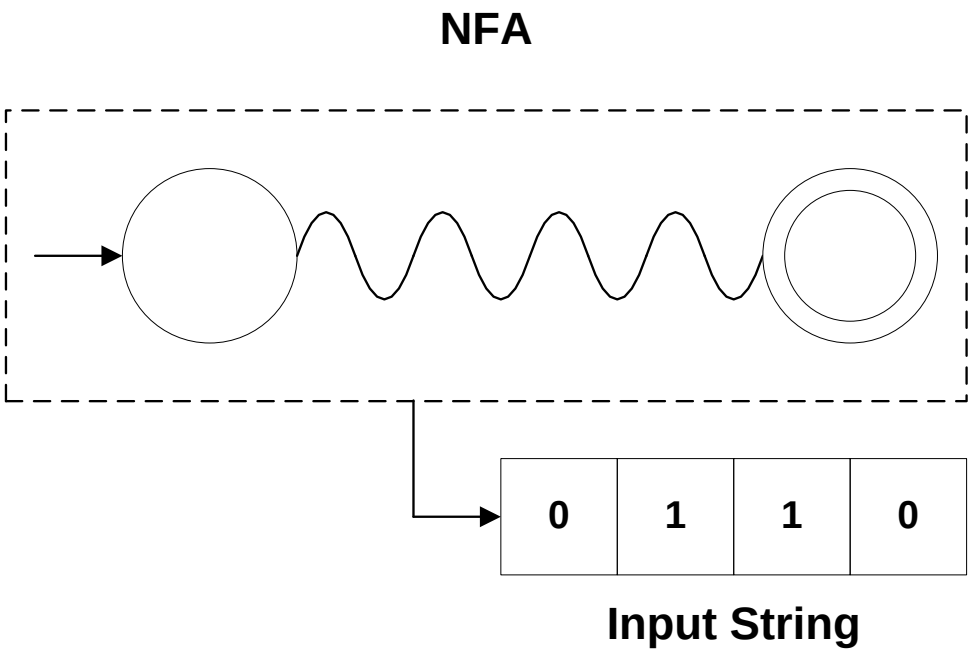
  It is used to determine whether a string belongs to a certain language or not. It is therefore a ***recognizing mechanism*** for regular languages.

- A ***pushdown automaton*** (PDA) is a machine that can recognize context-free languages.

INTRODUCTION TO PDAs

- A PDA is similar to an NFA except that it has access to a stack.

- Representation of an NFA:



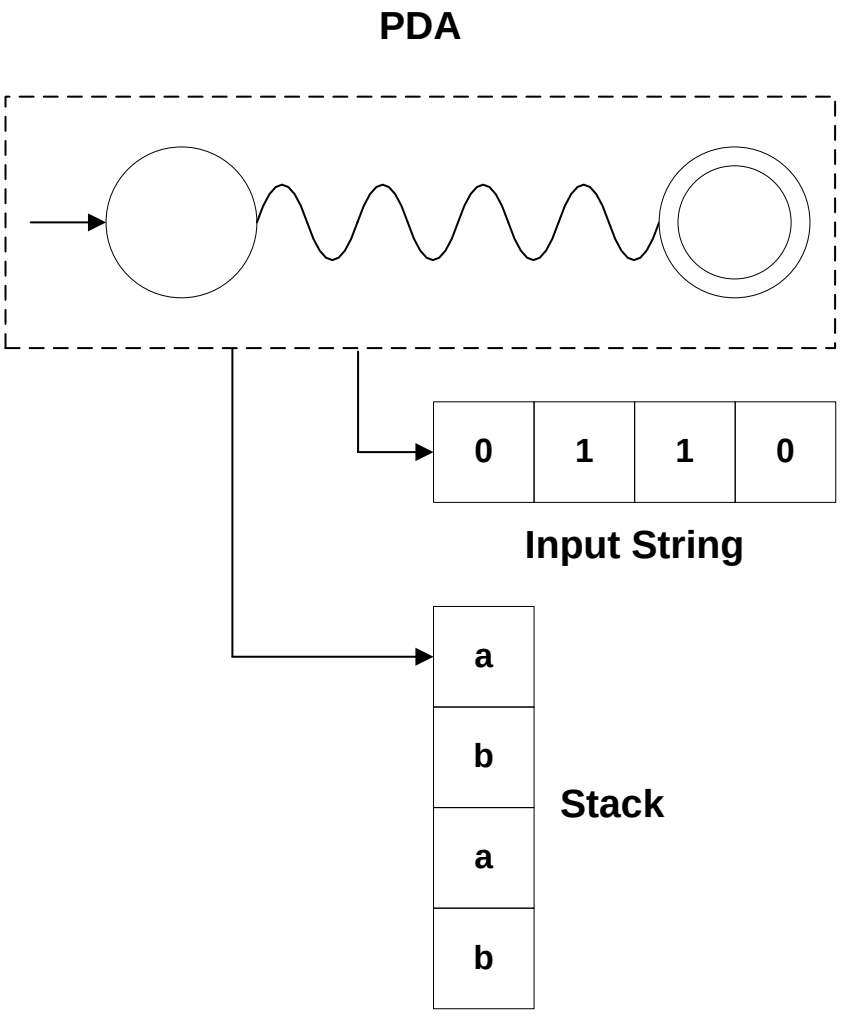
So, the NFA needs only the input string to perform its calculations

INTRODUCTION TO PDAs

- Representation of a PDA

  A PDA is similar to an NFA in the sense that it has the same components, which are a set of states, an input alphabet, a transition function, a start state, and a set of final states.

  The only difference is the addition of a sixth component which is a stack (also called a ***pushdown stack***).



INTRODUCTION TO PDAs

- The stack follows a last-in, first-out (LIFO) structure.
- Operations that can be done on the stack are ***push*** (add a symbol to the top of the stack) and ***pop*** (remove a symbol from the top of the stack).
- State transitions in a PDA depend on the current state, the incoming input symbol, and the symbol at the top of the stack.
- The stack can give additional memory beyond what is available in an NFA (as represented by the states).
- PDAs are more powerful than NFAs because they can recognize some non-regular languages.

INTRODUCTION TO PDAs

- Case Study 1:

  Consider the non-regular language $L_1 = \{0^n1^n \mid n \geq 0\}$. There is no NFA that can recognize this language but a PDA can be constructed to recognize it.

  PDA Operation:

  1. Read each symbol of the input string.
  2. While the input is a 0, push it onto the stack.
  3. While the input symbol is a 1, pop a 0 off the stack.
  4. If there is no more input symbol and the stack is empty, the input string is accepted.

FORMAL DEFINITION OF A PUSHDOWN AUTOMATON

- A pushdown automaton is a 6-tuple (Q, Σ, Γ, δ, q_o, 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. δ is the transition function,
5. $q_o \in Q$ and is the start state, and
6. $F \subseteq Q$ and is the set of final states.

FORMAL DEFINITION OF A PUSHDOWN AUTOMATON

- The stack alphabet Γ is the set of symbols that can be pushed onto the stack. It may or may not be the same as the input string alphabet Σ.

- Transitions in a PDA

  In an NFA, transitions are represented by:

$$\delta (q_i, o) = q_j$$

  This equation indicates that if the current state is state q_i and the input is o, the NFA goes to state q_j. State transitions are therefore determined by the current state and the input symbol.

FORMAL DEFINITION OF A PUSHDOWN AUTOMATON

- In a PDA, transitions are represented by:

$$\delta (q_i, o, a) = (q_j, b)$$

This equation indicates that if the current state is state q_i and the input symbol is o, and the symbol at the top of the stack is *a*, the PDA goes to state q_j and pushes the symbol *b* onto the stack.

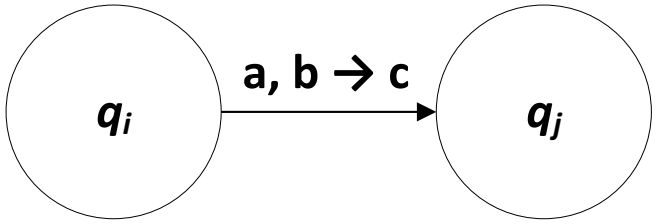
State transitions are thus determined by the current state, the input symbol, and the symbol at the top of the stack.

PDA STATE DIAGRAMS

- PDA State Transitions

  PDA state diagrams are similar to NFA state diagrams except for the labels of the transition edges.

  Example:



The label $a, b \rightarrow c$ means that if the input symbol is a and the symbol read from the top of the stack is b, then the PDA goes to state q_j and it pushes the symbol c unto the stack.

PDA STATE DIAGRAMS

- Take note that reading the symbol at the top of the stack implies a pop operation. The label $a, b \rightarrow c$ implies that the symbol b was popped off the stack.

- Furthermore, any of the symbols a, b, c in the given label can also be the empty string ε.

1. If $a = \varepsilon$, the PDA makes the transition without any input symbol (PDAs are nondeterministic).
2. If $b = \varepsilon$, the PDA makes the transition without reading any symbol from the stack.
3. If $c = \varepsilon$, the PDA does not push any symbol onto the stack.

PDA STATE DIAGRAMS

- In order for the PDA to determine if the stack is empty, a special symbol will be designated as the stack empty symbol.

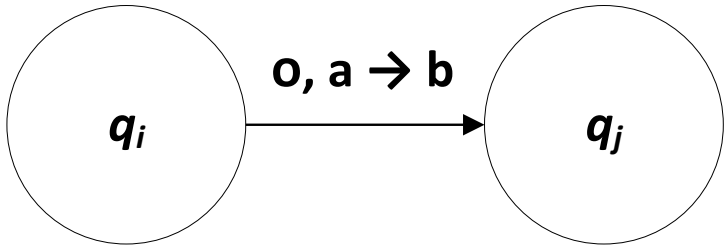
  This symbol will be pushed onto the stack every time a computation is started.

  The symbol that will be used here will be the dollar sign ($).

  So, any time the PDA sees that the symbol at the top of the stack is $, then the stack is considered empty.

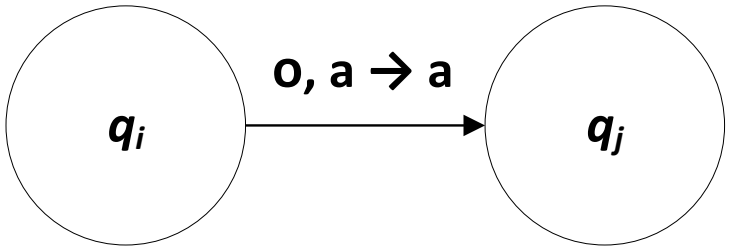
PDA STATE DIAGRAMS

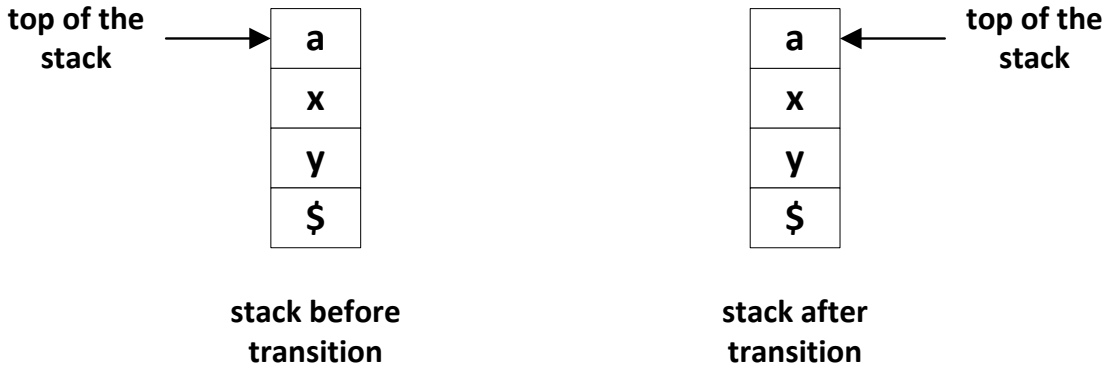
- Sample Transitions:



PDA STATE DIAGRAMS

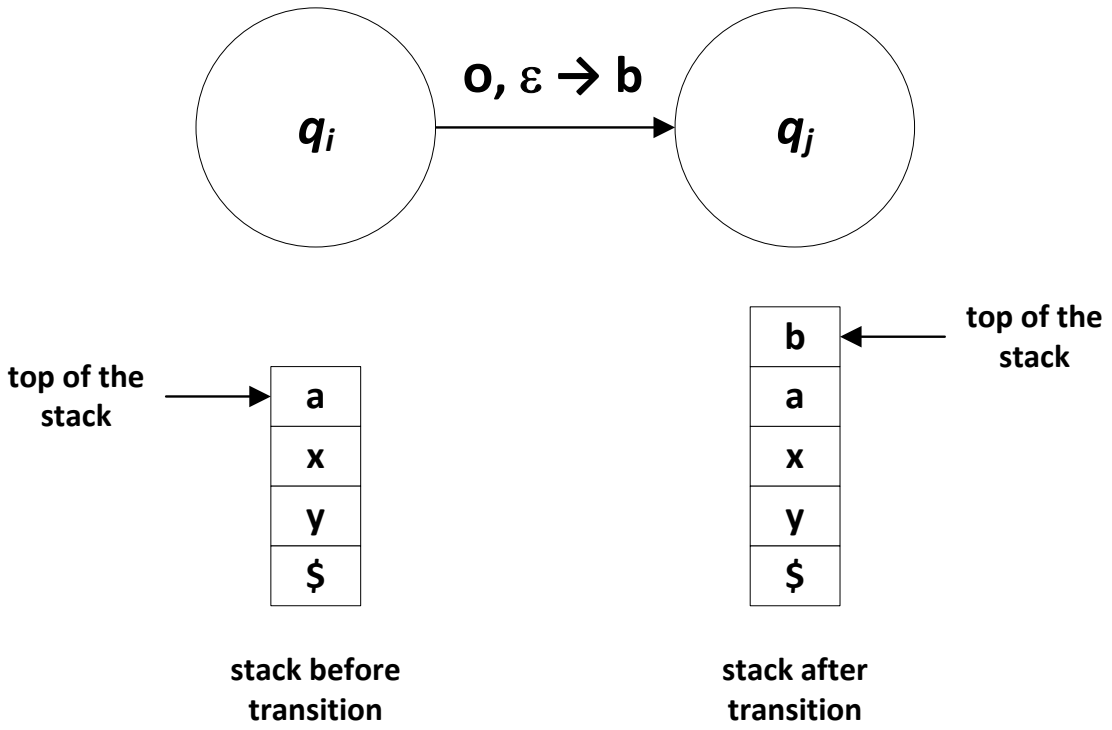
- Sample Transitions:





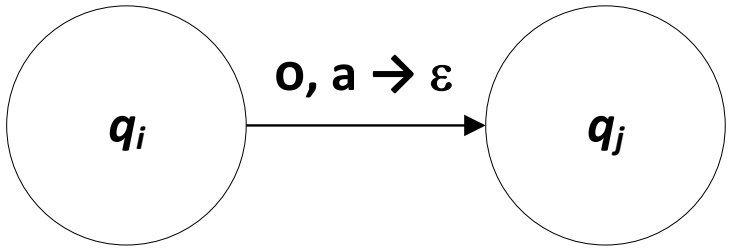
PDA STATE DIAGRAMS

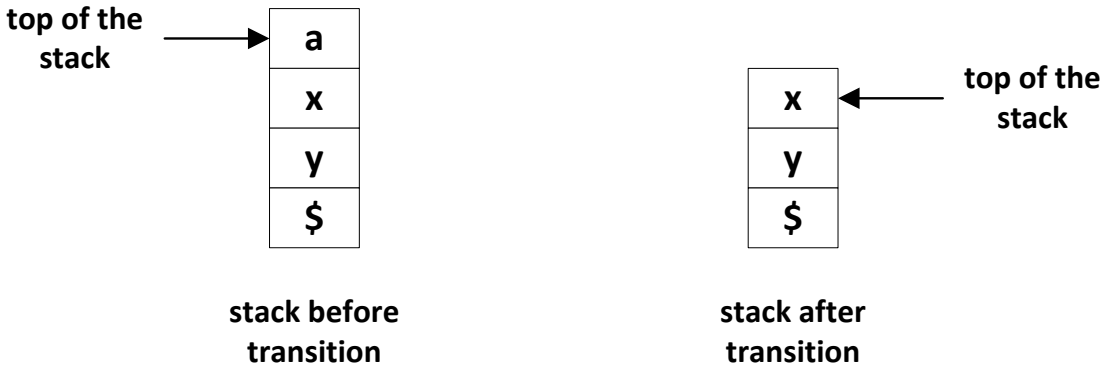
- Sample Transitions:



PDA STATE DIAGRAMS

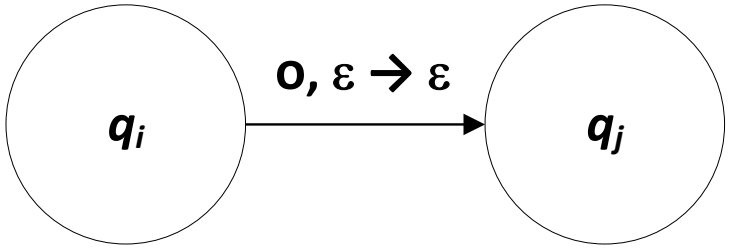
- Sample Transitions:

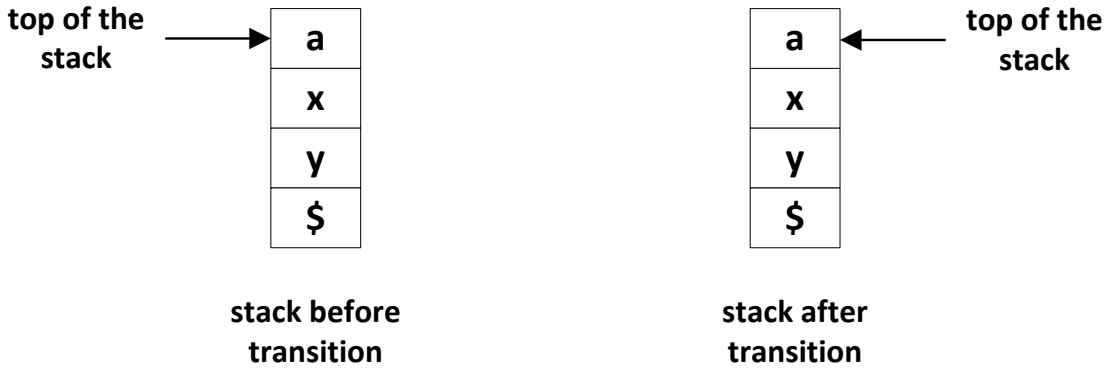




PDA STATE DIAGRAMS

- Sample Transitions:

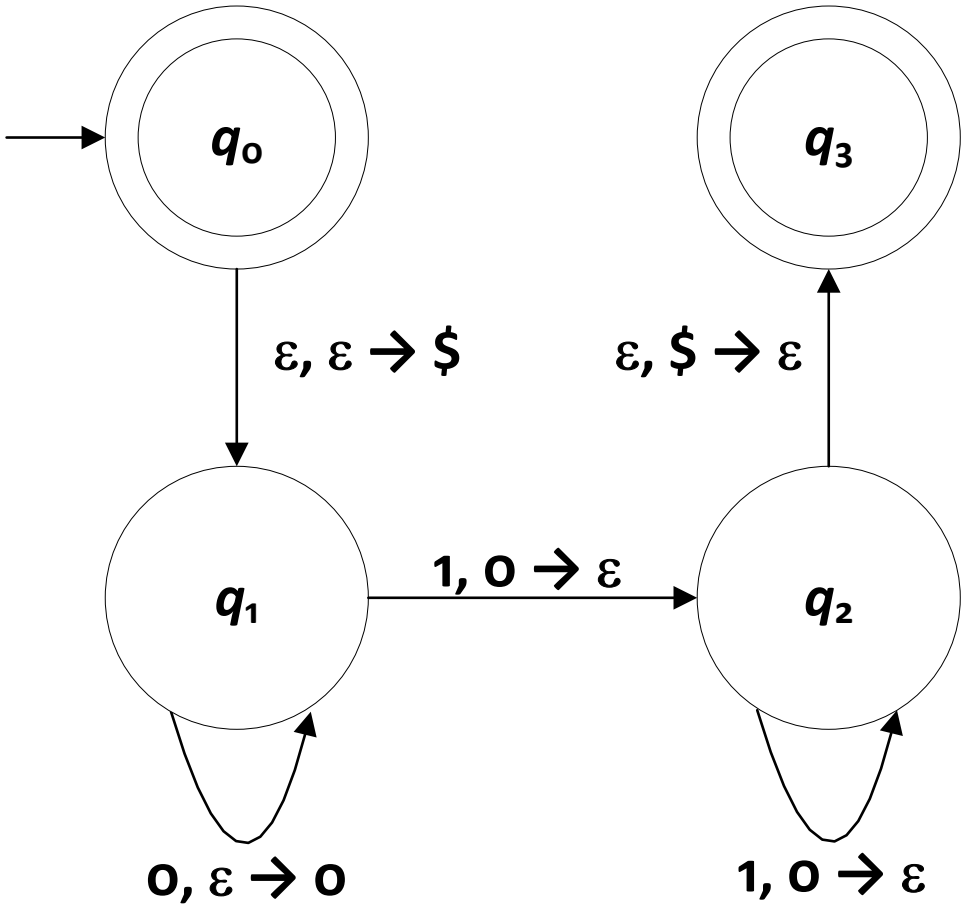




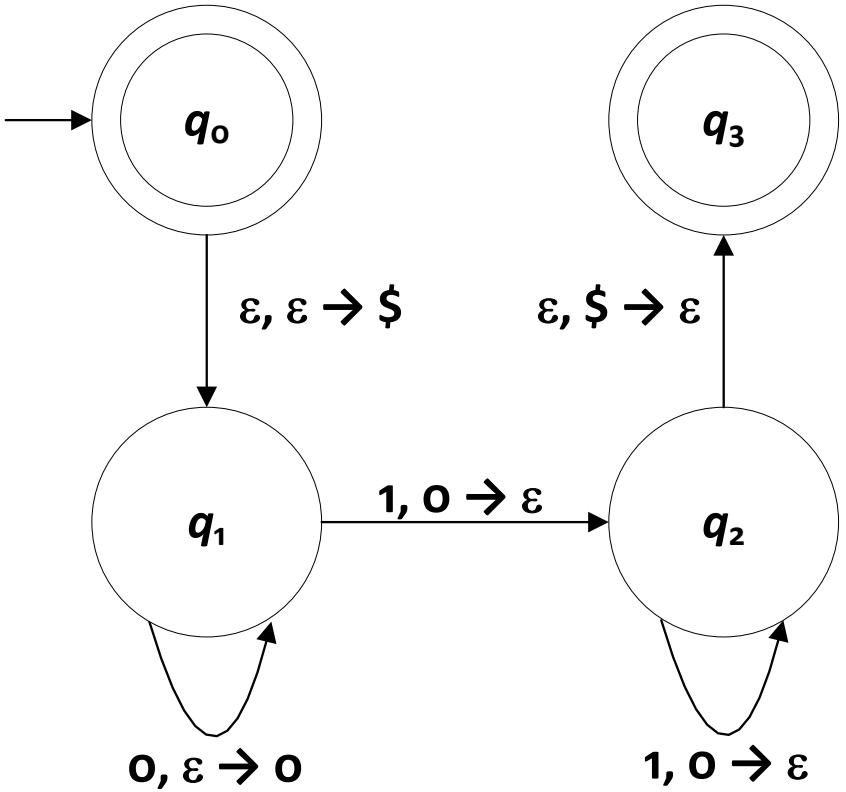
PDA STATE DIAGRAMS

- Continuation of Case Study 1

  The PDA P_1 that can recognize the non-regular language $L_1 = \{0^n1^n \mid n \geq 0\}$ is:



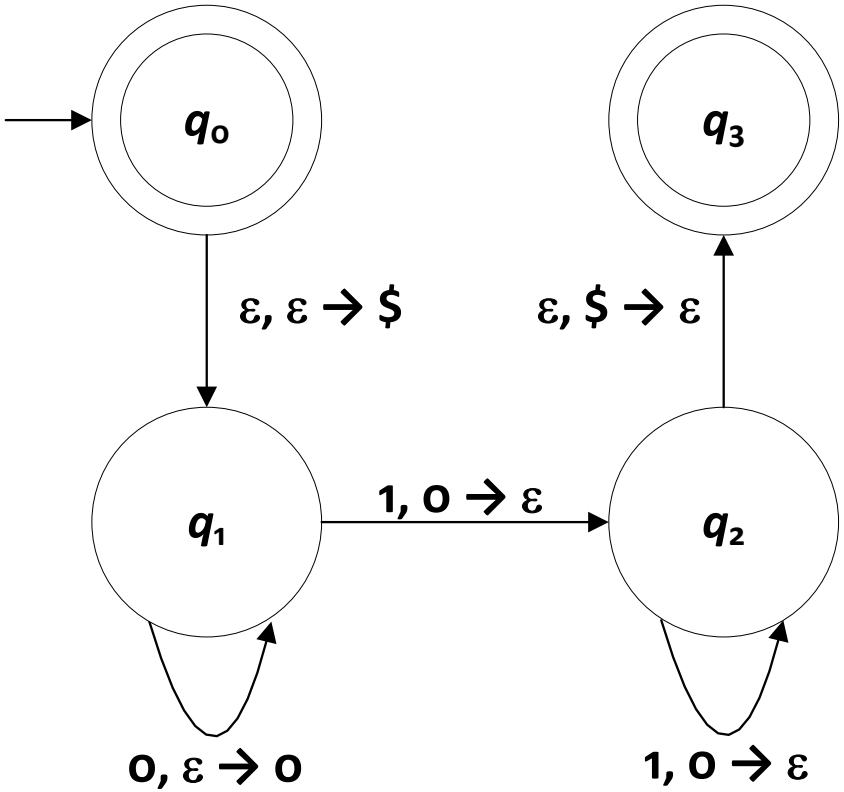
PDA STATE DIAGRAMS



The given PDA can be formally defined as $P_1 = (Q, \Sigma, \Gamma, \delta, q_o, F)$ where:

1. $Q = \{q_o, q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. $\Gamma = \{0, \$\}$
4. The start state is q_o.
5. $F = \{q_o, q_3\}$

PDA STATE DIAGRAMS



6. The transition function δ is given by:

Input	Stack	Current State			
		q_0	q_1	q_2	q_3
0	0				
	$				
	ε		$(q_1, 0)$		
1	0		(q_2, ε)	(q_2, ε)	
	$				
	ε				
ε	0				
	$			(q_3, ε)	
	ε	$(q_1, \$)$			

PDA STATE DIAGRAMS

- PDA P_1 Operation

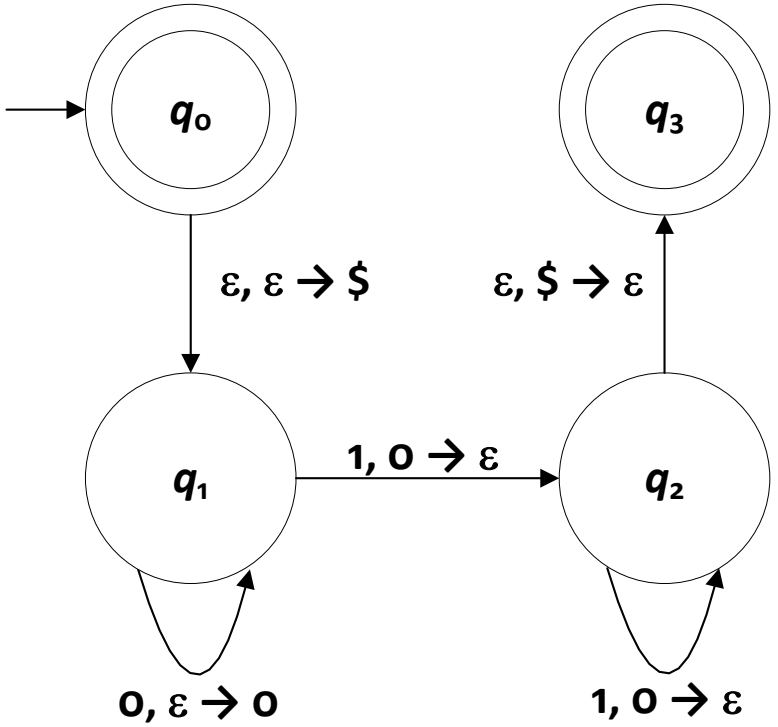
1. Before any computation begins, the PDA will be at the start state q_o.

2. There is a transition edge from state q_o to state q_1 with the label **ε, ε → $**.

   Without considering the input symbol and the symbol at the top of the stack, the PDA may opt to go to state q_1 and at the same time pushing the *$* symbol onto the stack.

   This pushes the *$* symbol onto the stack since it should always the first symbol to enter the stack.

   All PDAs will start in this manner.



PDA STATE DIAGRAMS

- PDA P_1 Operation

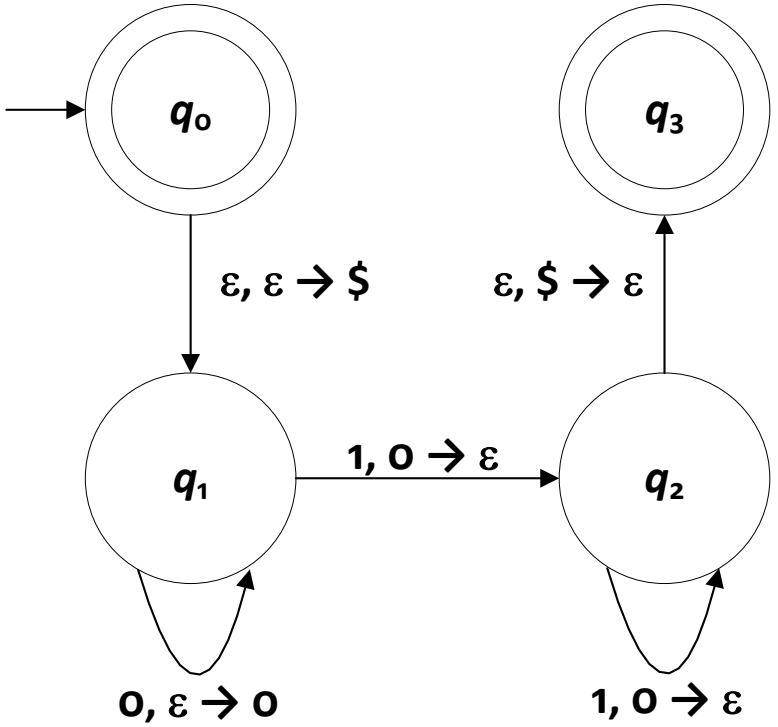
3. State q_1 is the state where the PDA starts pushing 0s onto the stack.

   At this state, there are two transition edges.

   The first has the label **0, ε → 0** and goes back to state q_1.

   When a 0 arrives at the input, the PDA will stay at state q_1 and at the same time push a 0 onto the stack.

   While 0s keep on arriving at the input, the PDA will simply push 0s onto the stack (and stay at state q_1).



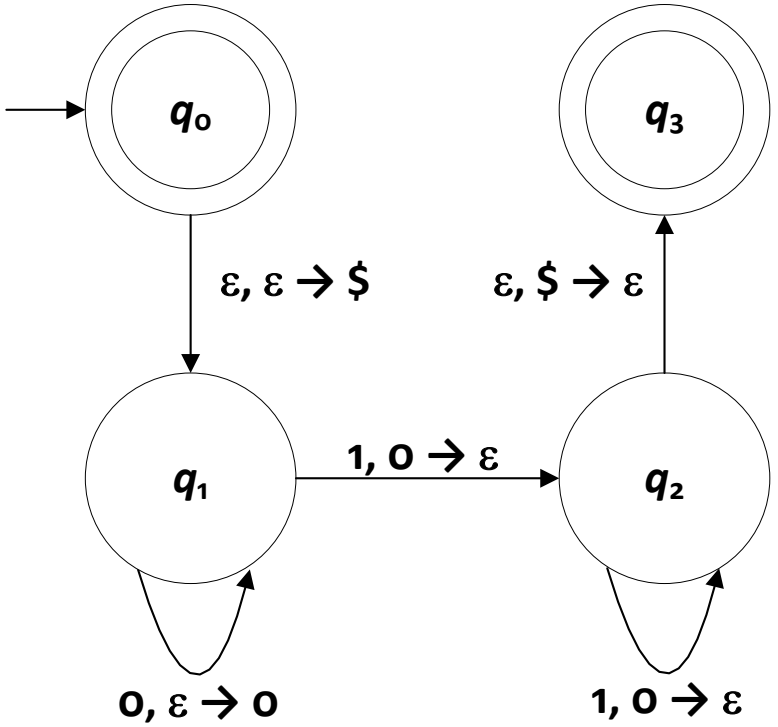
PDA STATE DIAGRAMS

- PDA P_1 Operation

  The other edge has a label **1, 0 → ε** which leads to state q_2 (the state where the PDA starts popping 0s and matching them with incoming 1s).

  When a 1 arrives and the symbol at the top of the stack is 0, the PDA goes to q_2 without pushing anything onto the stack.

  What happened here is that the first 1 that arrived has been partnered with a 0 that is at the top of the stack.



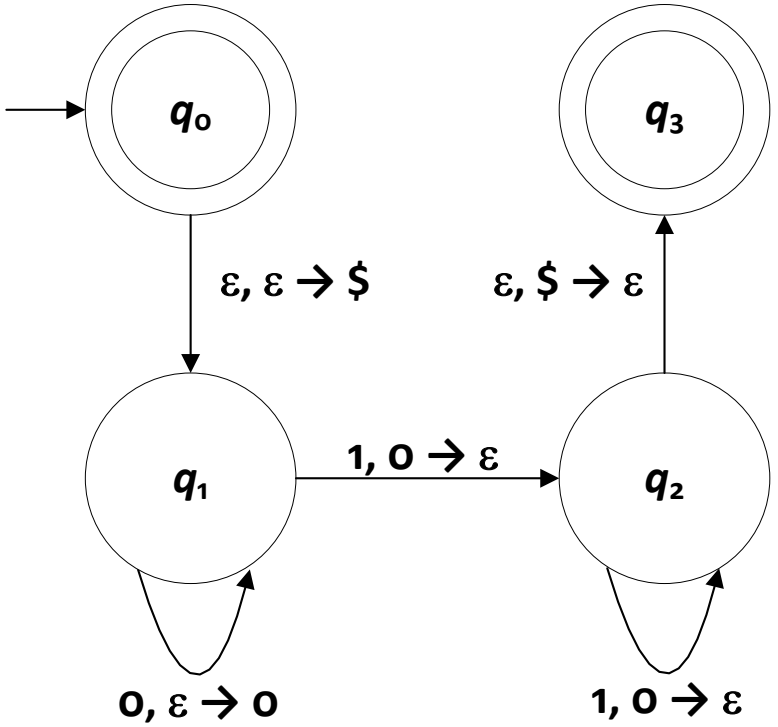
PDA STATE DIAGRAMS

- PDA P_1 Operation

  The PDA will then go to state q_2 where it will start partnering each succeeding 1 it will be receiving with the 0s it has received by popping a 0 off the stack for each 1 it receives.

  If the PDA receives a 1 while it is at state q_1 and the symbol at top of the stack is not a 0, this means that the PDA received a 1 without receiving any 0s yet.

  So PDA cannot go anywhere and the computation dies (the string is rejected).



PDA STATE DIAGRAMS

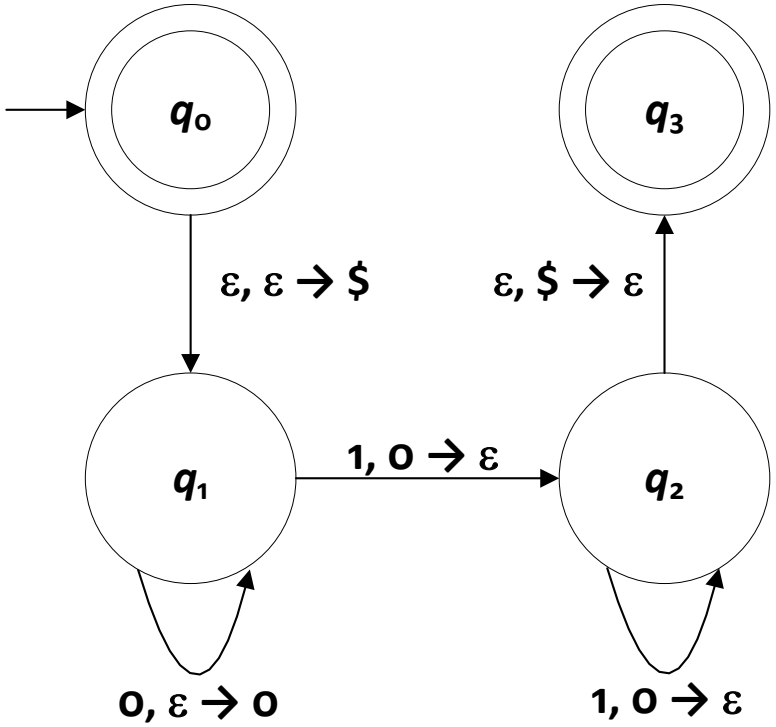
- PDA P_1 Operation

4. At state q_2, there are also two transition edges.

   One edge has the label **1, 0 → ε** and goes back to state q_2.

   As mentioned earlier, this is the point where the 1s arrive. For every 1 that arrives, the PDA pops a 0 from the stack and goes back to state q_2.

   Take note that nothing is pushed onto the stack at this point of the computation.



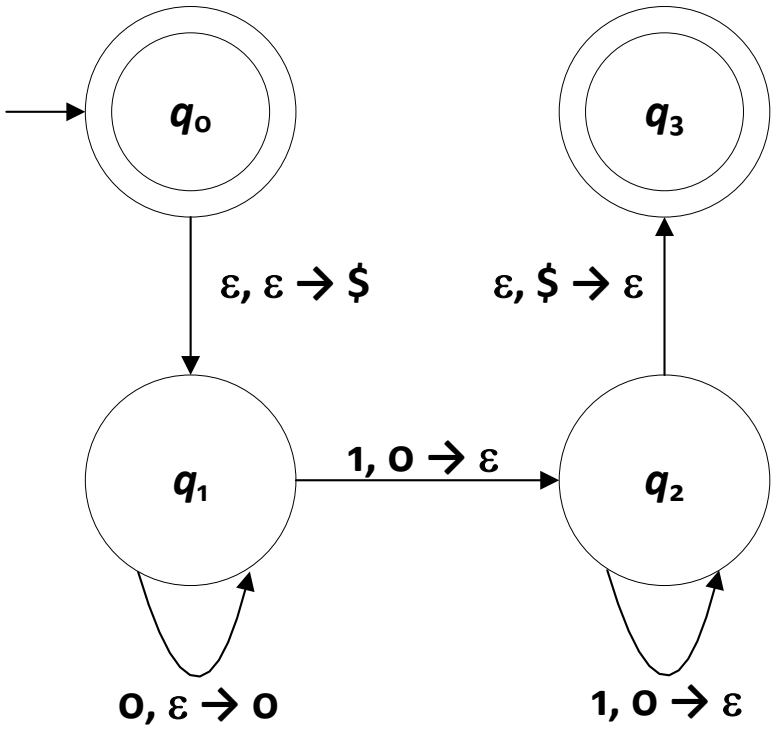
PDA STATE DIAGRAMS

- PDA P_1 Operation

  If a 0 suddenly arrives at the input, it is an out-of-sequence 0 (a 0 arriving after a 1) so the computation dies (the string is rejected).

  If a 1 arrives and the symbol at the top of the stack is not 0, this means that the stack ran out of 0s.

  This means that there are more 1s than 0s. Once again, the computation dies (the string is rejected).



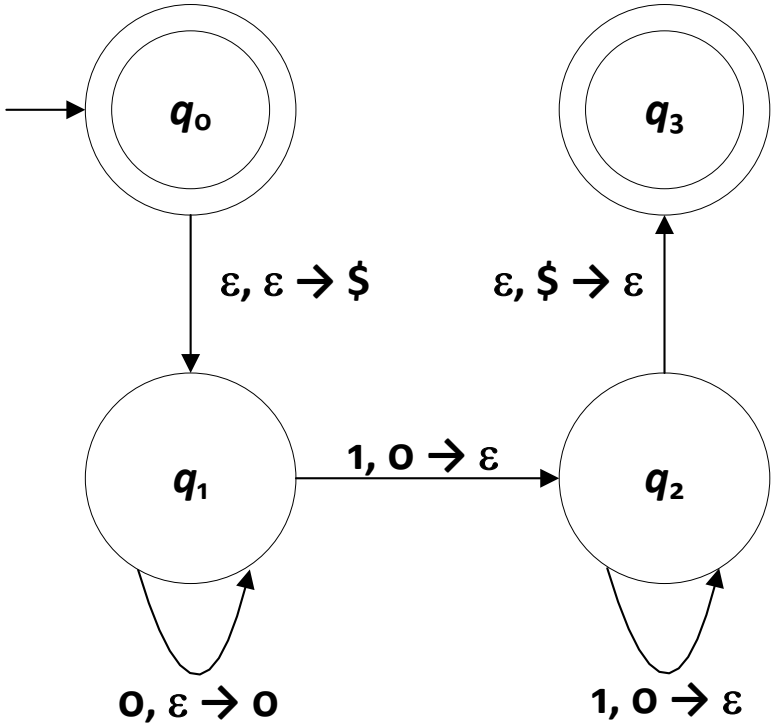
PDA STATE DIAGRAMS

- PDA P_1 Operation

The other transition edge has the label **ε, $ → ε** which leads to state q_3 (a final state).

This means that if no input symbol arrives and the symbol at the top of the stack is $ (the stack is empty), then this indicates that every 0 that was in the stack was partnered with each 1 that arrived at the input.

The PDA then goes to state q_3 which is a final state and the string is accepted.



PDA STATE DIAGRAMS

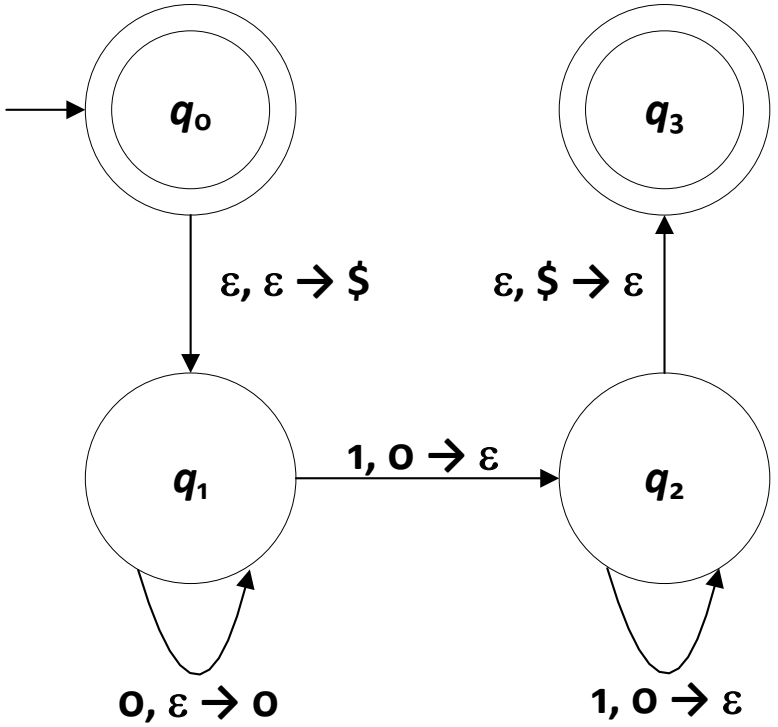
- PDA P_1 Operation

  If a 1 arrives and the symbol at the top of the stack is $, this means that the input that arrived is an excess 1.

  So the computation dies (the string is rejected).

  If a 0 arrives at this point of the computation, it is an out-of-sequence 0 (a 0 arriving after the 1s).

  The computation also dies (the string is rejected).



PDA STATE DIAGRAMS

- PDA P_1 Operation

5. At state q_3, there are no transition edges.

   A 0 that arrives at this point is an out-of-sequence 0 while a 1 that arrives at this point is an excess 1.

   So any input that arrives causes the computation to die and the string is rejected.