PUSHDOWN AUTOMATA

iACADEMY
SCHOOL OF COMPUTING • SCHOOL OF BUSINESS • SCHOOL OF DESIGN

SCHOOL OF COMPUTING
MITCH M. ANDAYA

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.

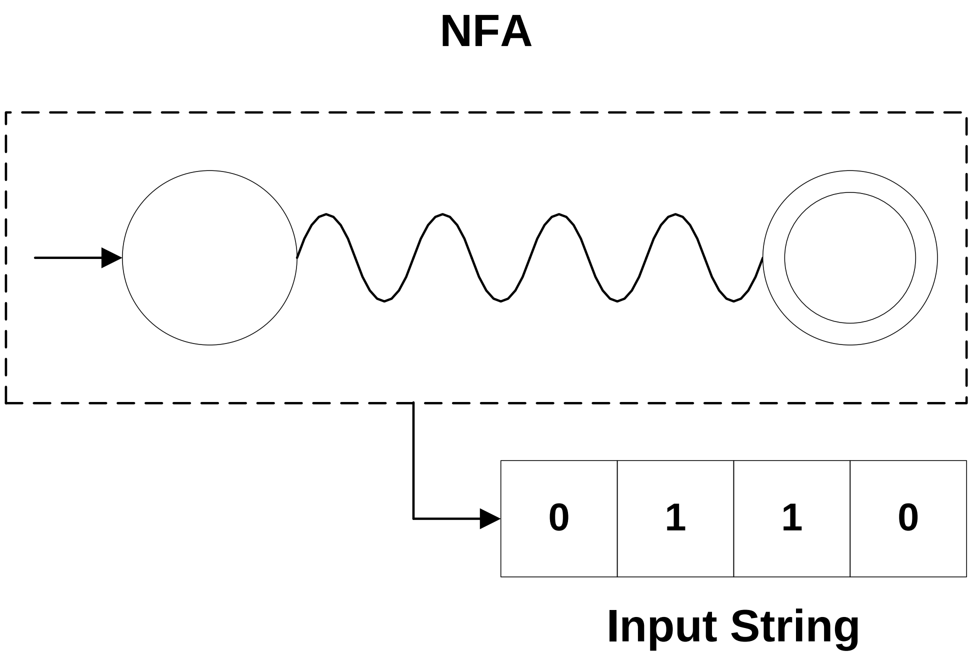
  It is used to determine whether a string belongs to a certain language or not. It is therefore a ***recognizing mechanism*** for regular languages.

- 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:



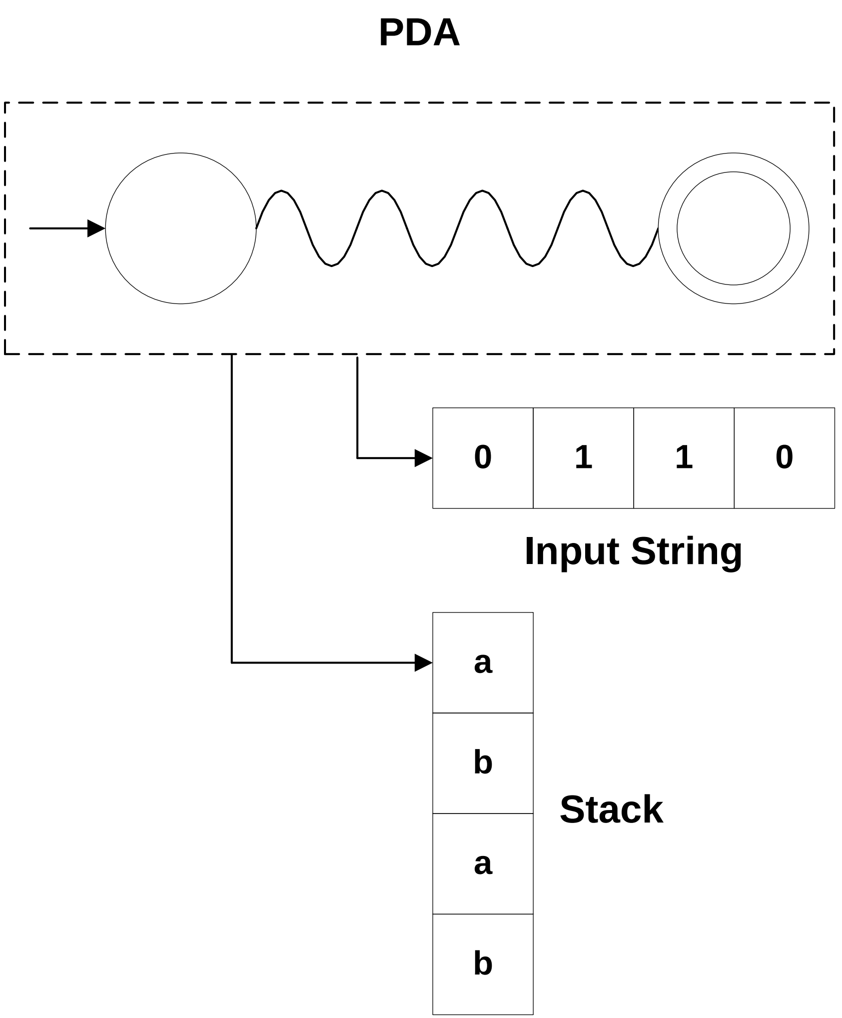
So, the NFA needs only the input string to perform its calculations

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.

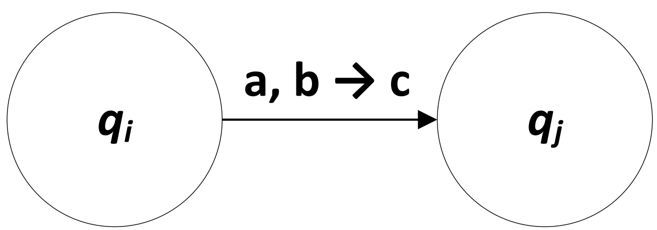
State transitions are thus determined by the current state, the input symbol, and the symbol at the top of the stack.

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.

iACADEMY
SCHOOL OF COMPUTING • SCHOOL OF BUSINESS • SCHOOL OF DESIGN

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.

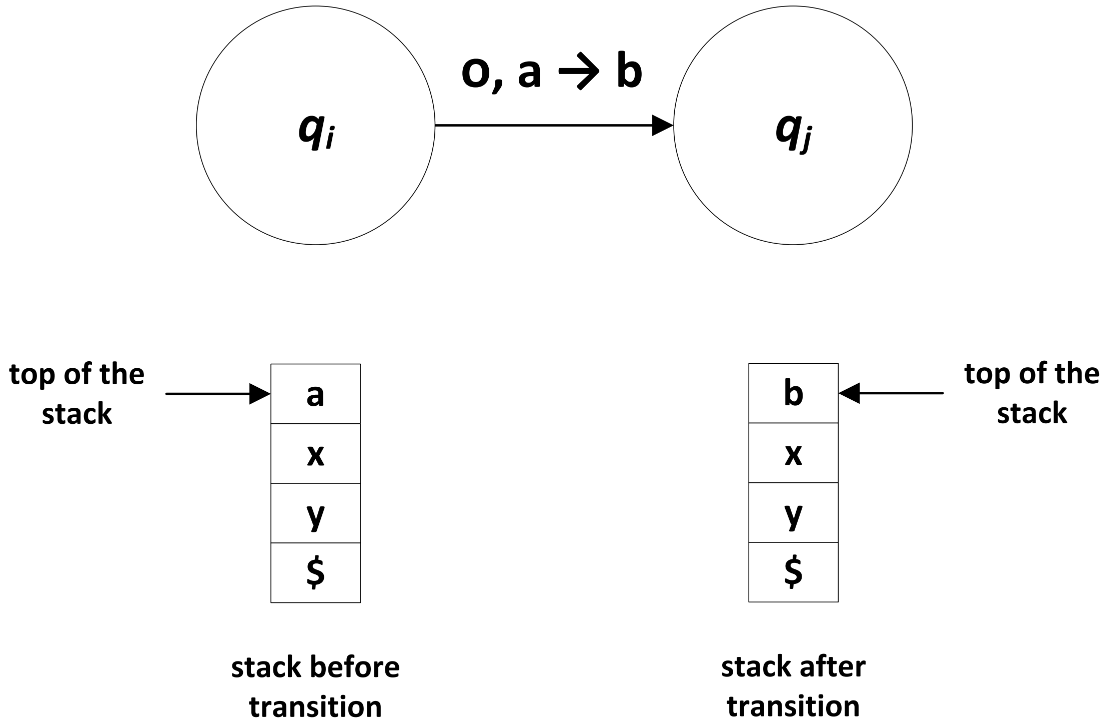
  This symbol will be pushed onto the stack every time a computation is started.

  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.

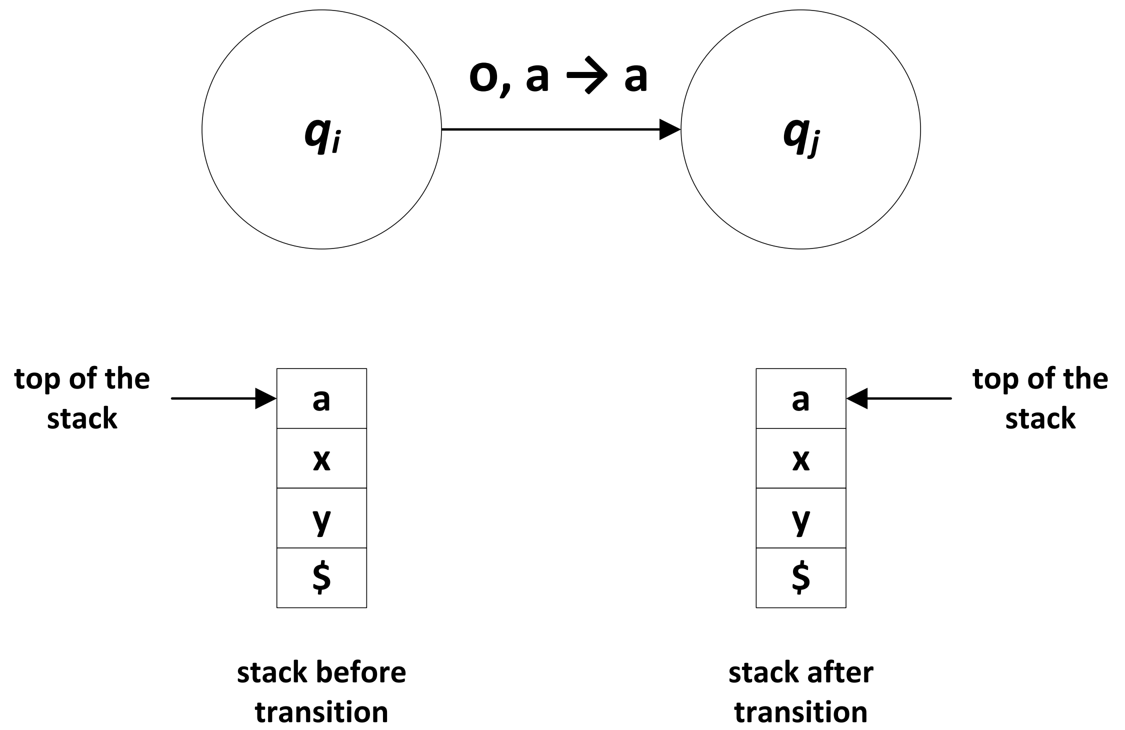
PDA STATE DIAGRAMS

- Sample Transitions:



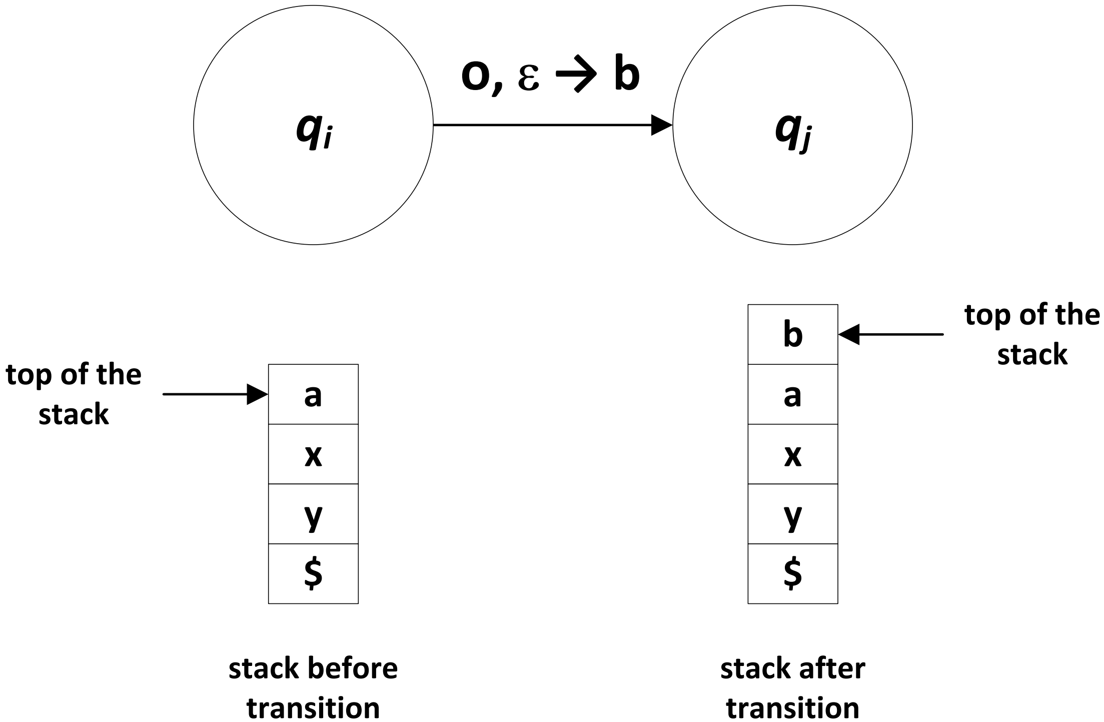
PDA STATE DIAGRAMS

- Sample Transitions:



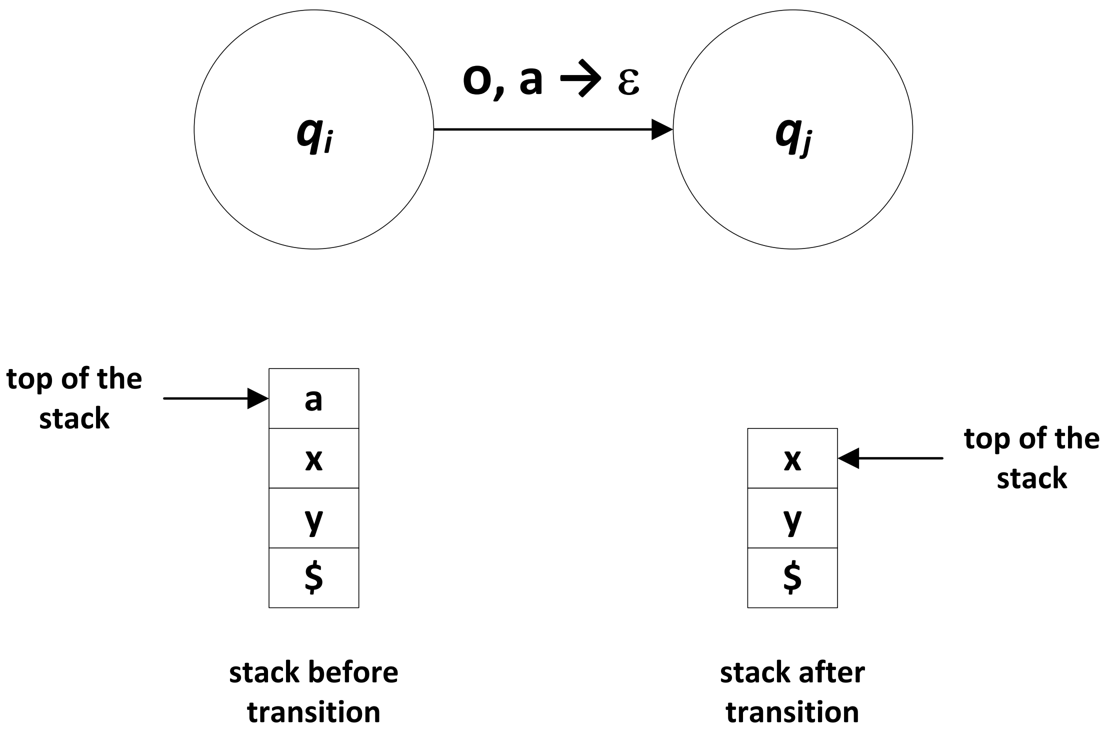
PDA STATE DIAGRAMS

- Sample Transitions:



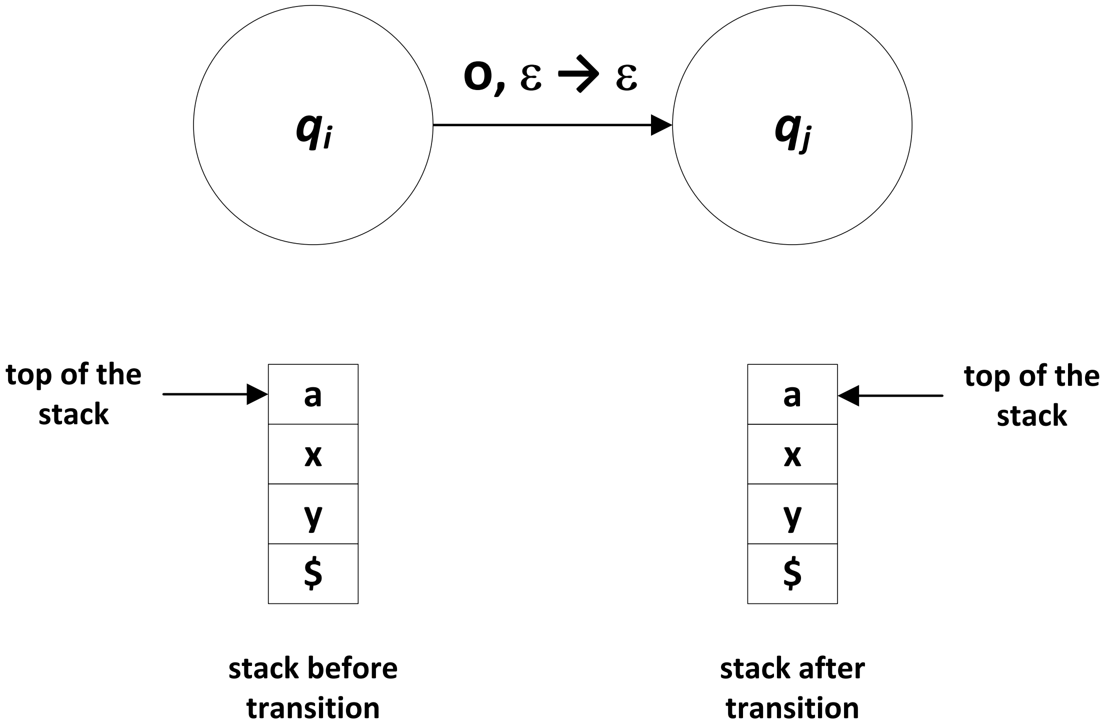
PDA STATE DIAGRAMS

- Sample Transitions:



PDA STATE DIAGRAMS

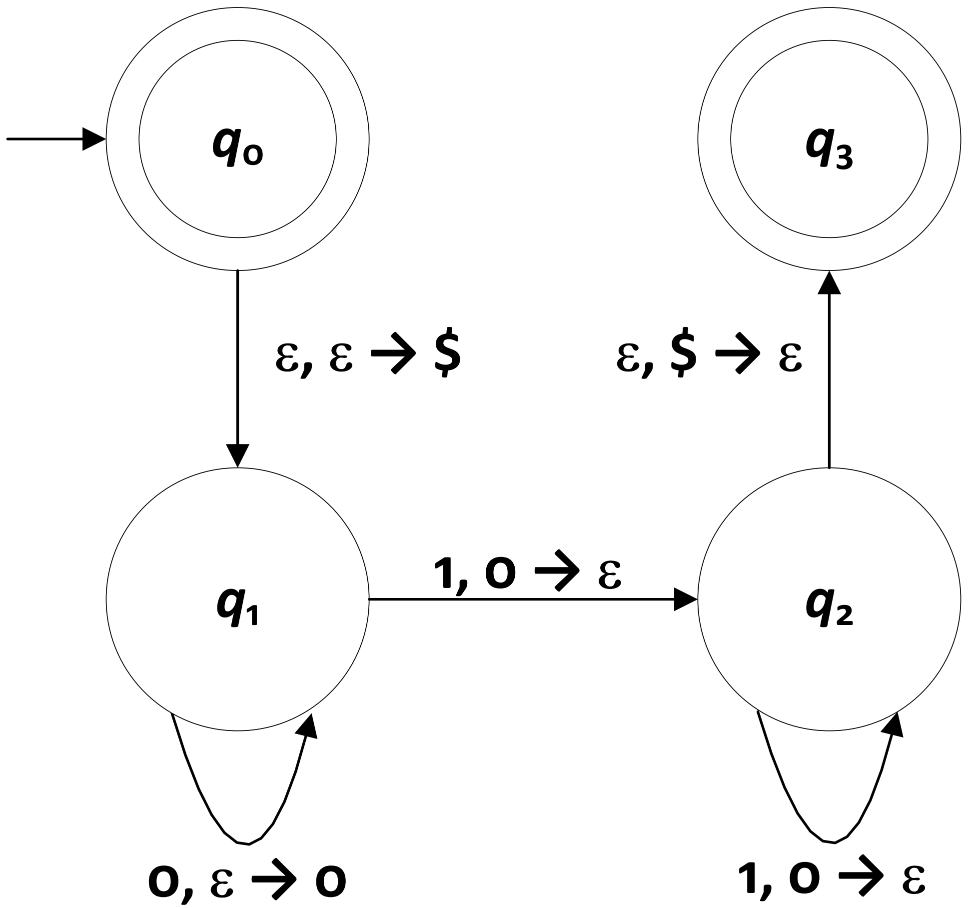
- Sample Transitions:



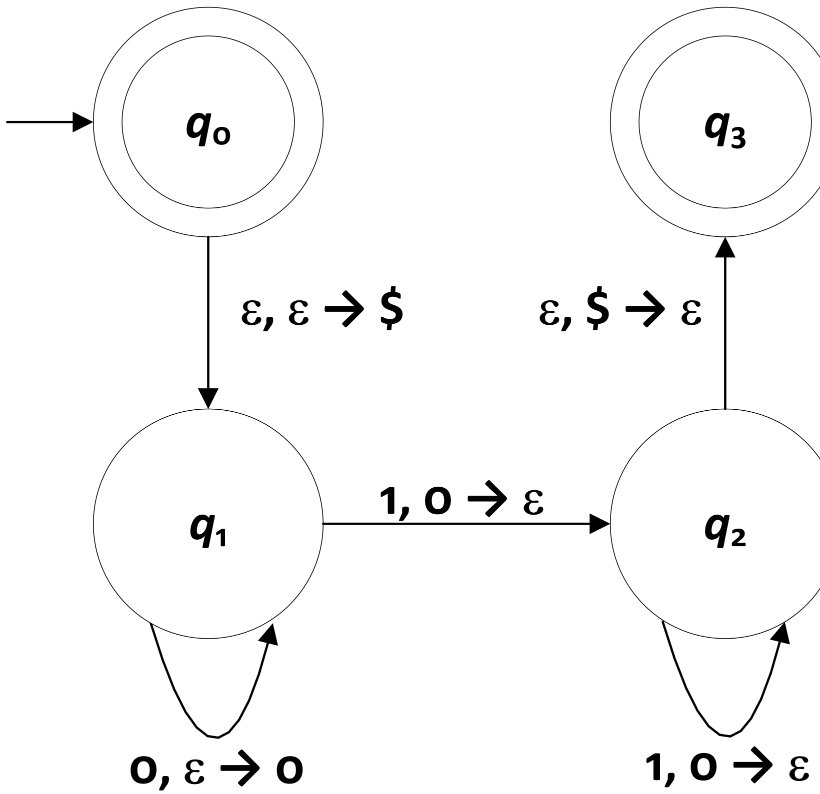
PDA STATE DIAGRAMS

- 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:



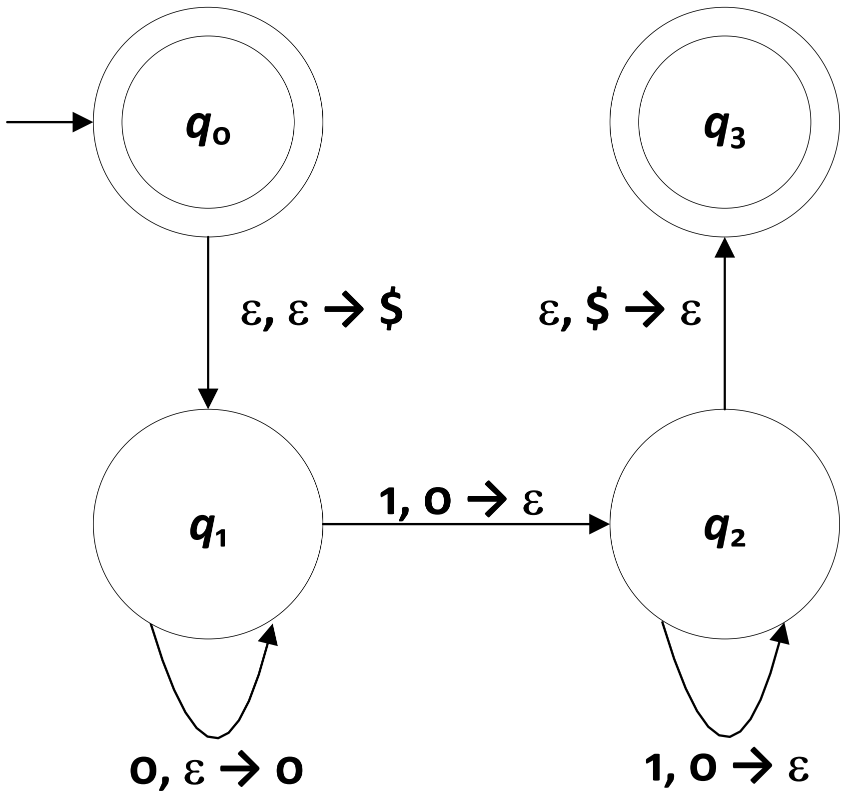
PDA STATE DIAGRAMS



The given PDA can be formally defined as $P_1 = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where:

1. $Q = \{q_0, q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. $\Gamma = \{0, \$\}$
4. The start state is q_0.
5. $F = \{q_0, q_3\}$

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

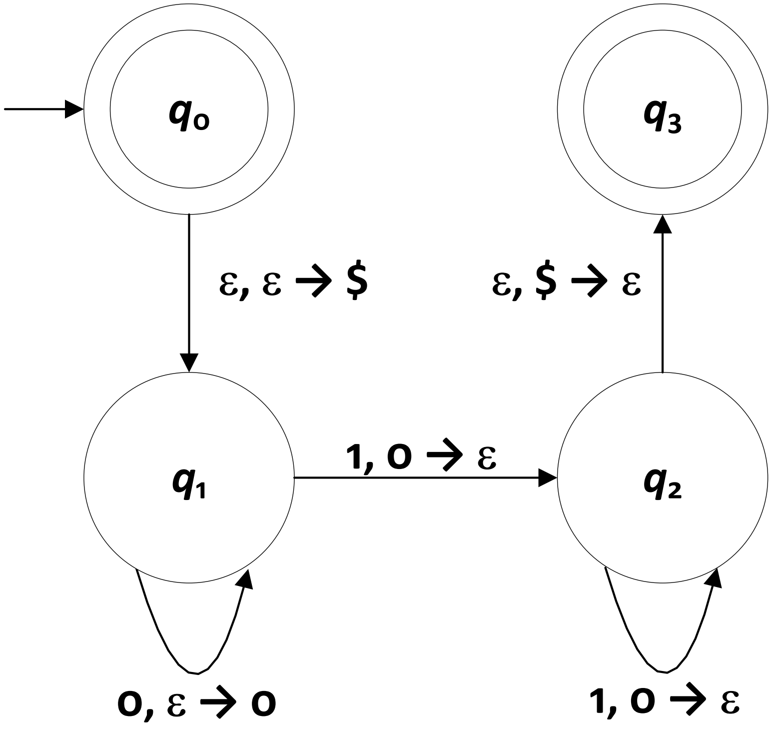
1. Before any computation begins, the PDA will be at the start state q_0.

2. There is a transition edge from state q_0 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

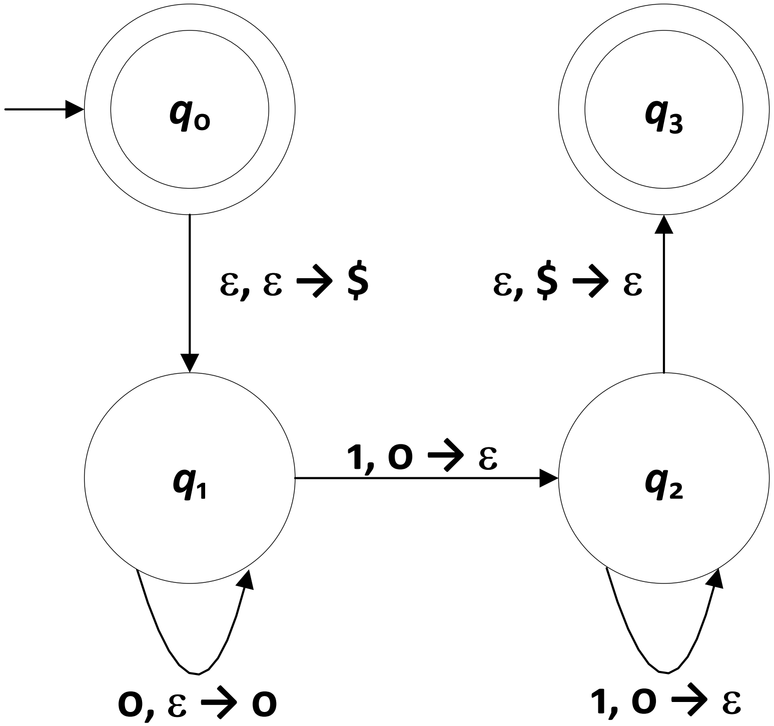
3. State q_1 is the state where the PDA starts pushing 0s onto the stack.

   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).



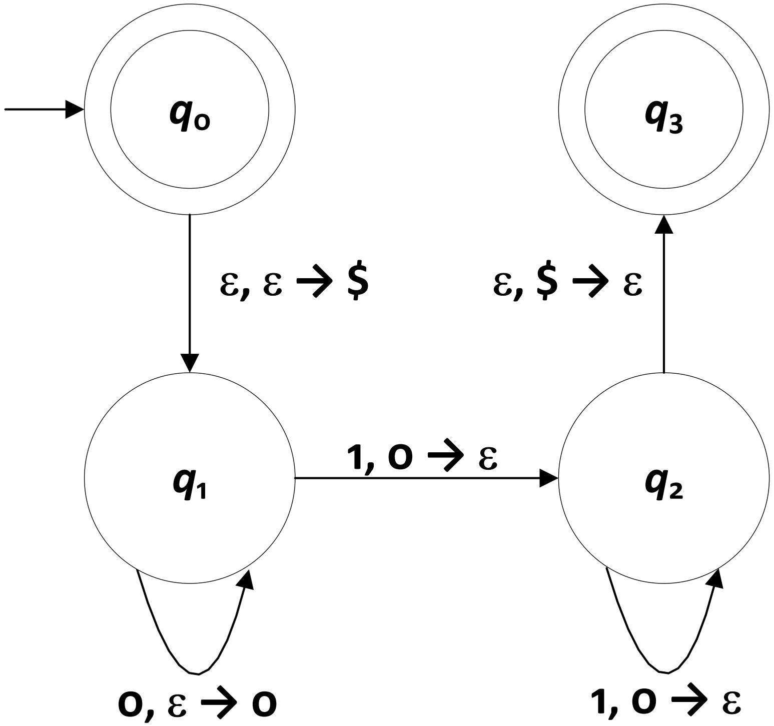
PDA STATE DIAGRAMS

- 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.



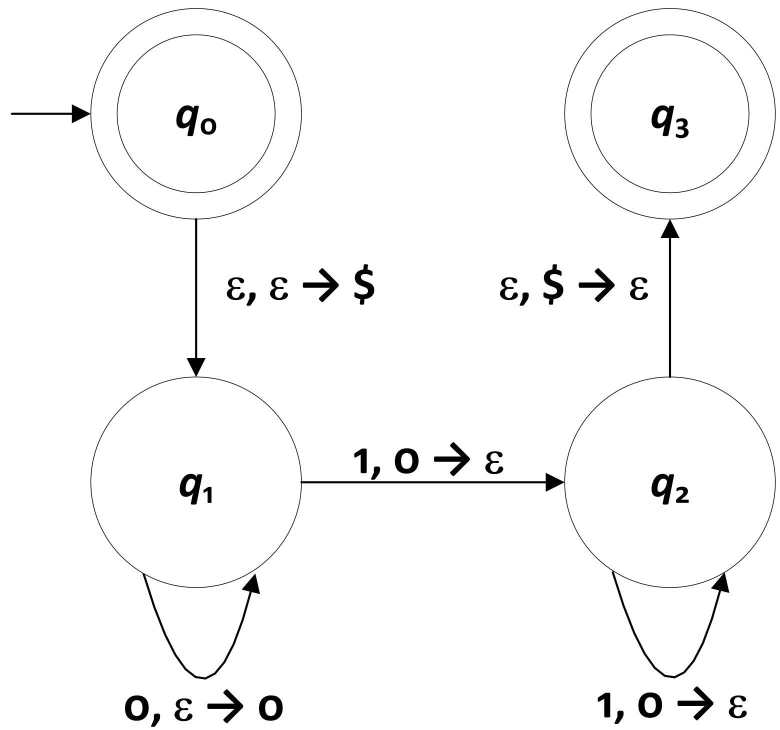
PDA STATE DIAGRAMS

- 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

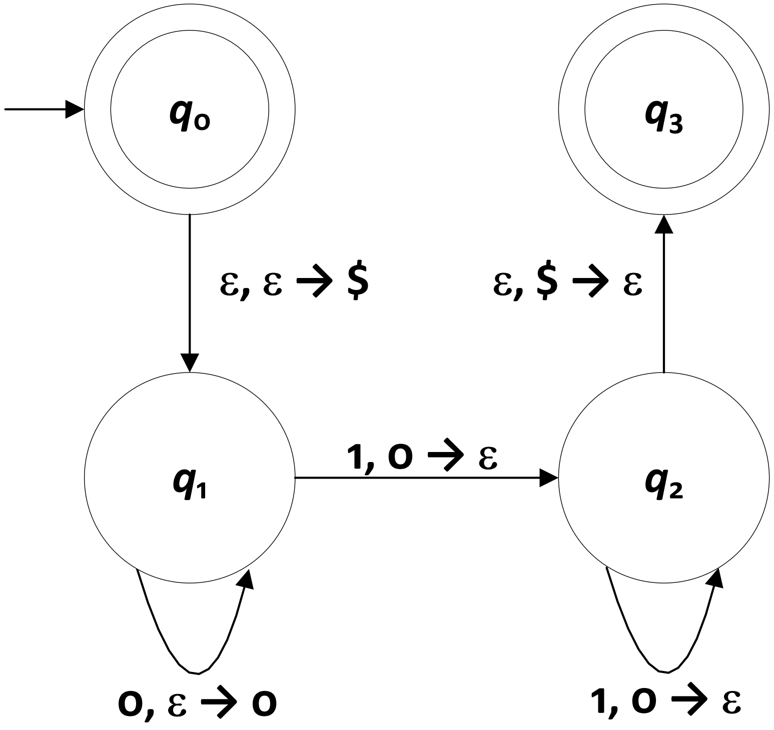
- PDA P_1 Operation

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.



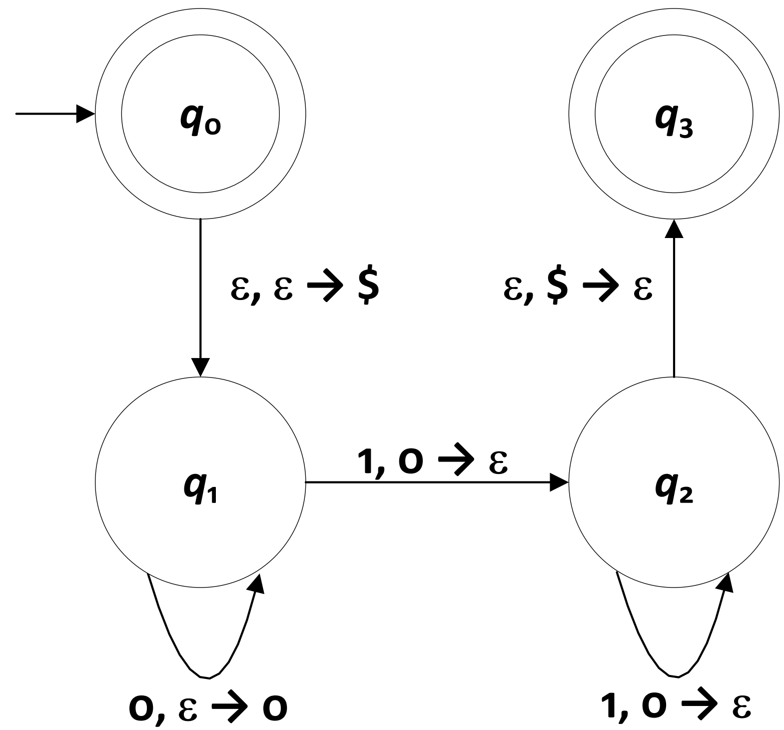
PDA STATE DIAGRAMS

- 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).



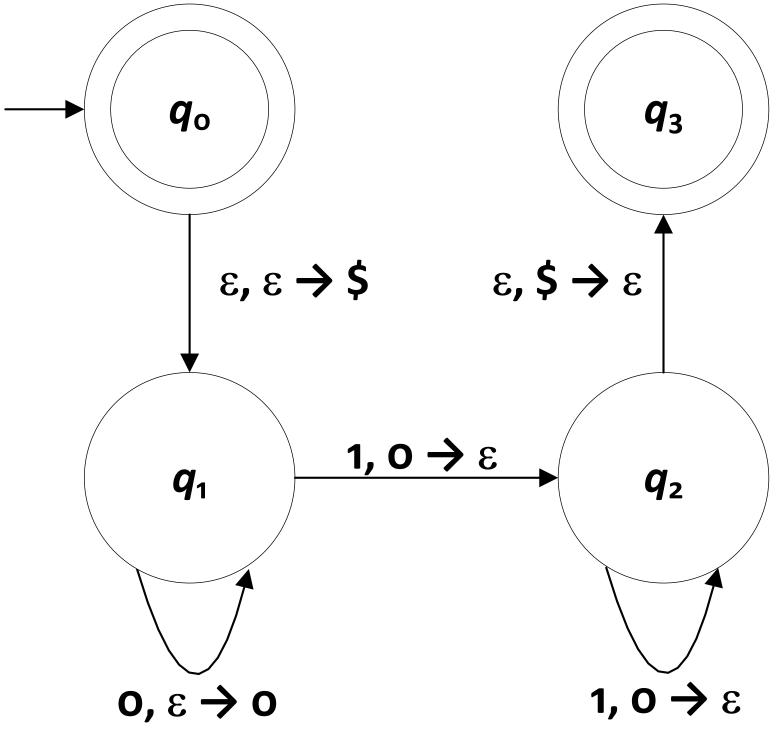
PDA STATE DIAGRAMS

- 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

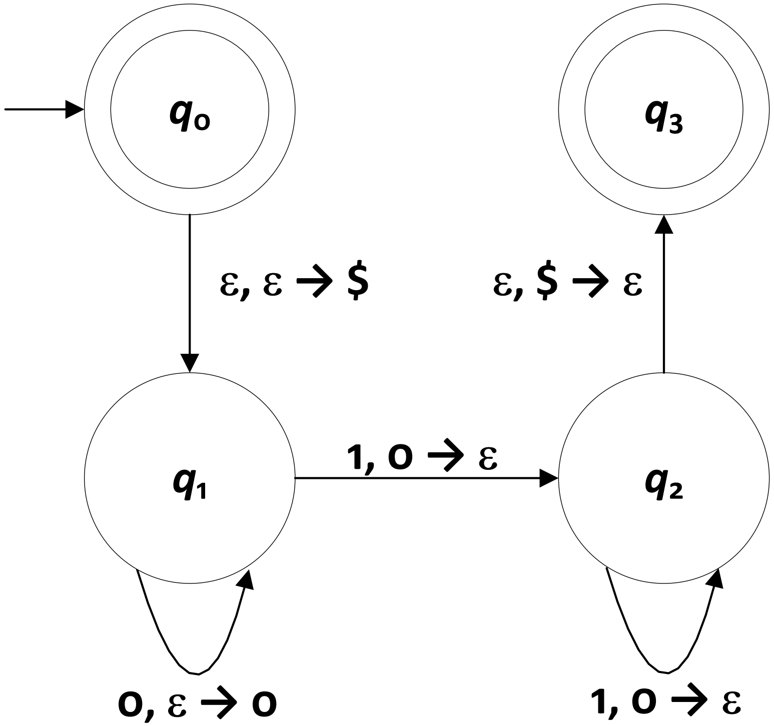
- PDA P_1 Operation

  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).



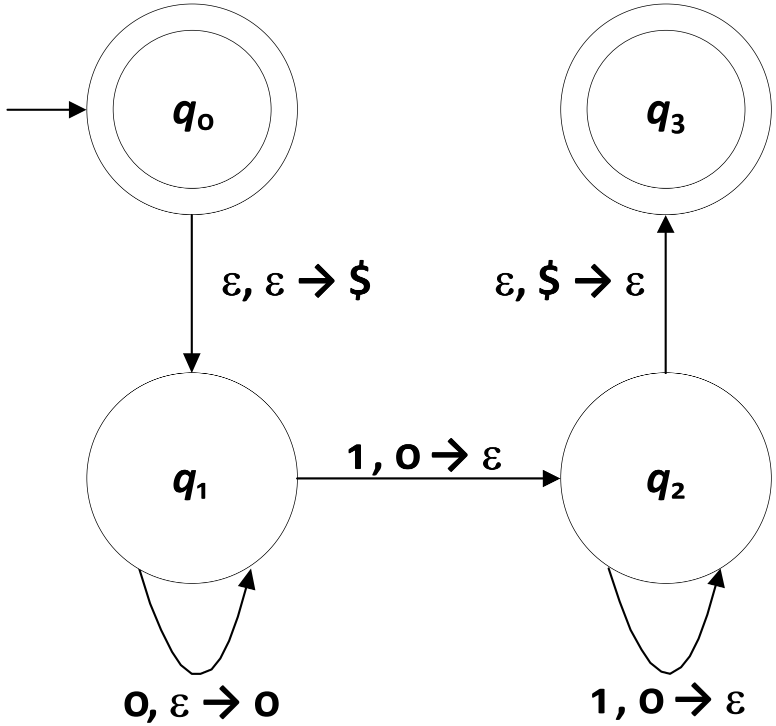
PDA STATE DIAGRAMS

- 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.

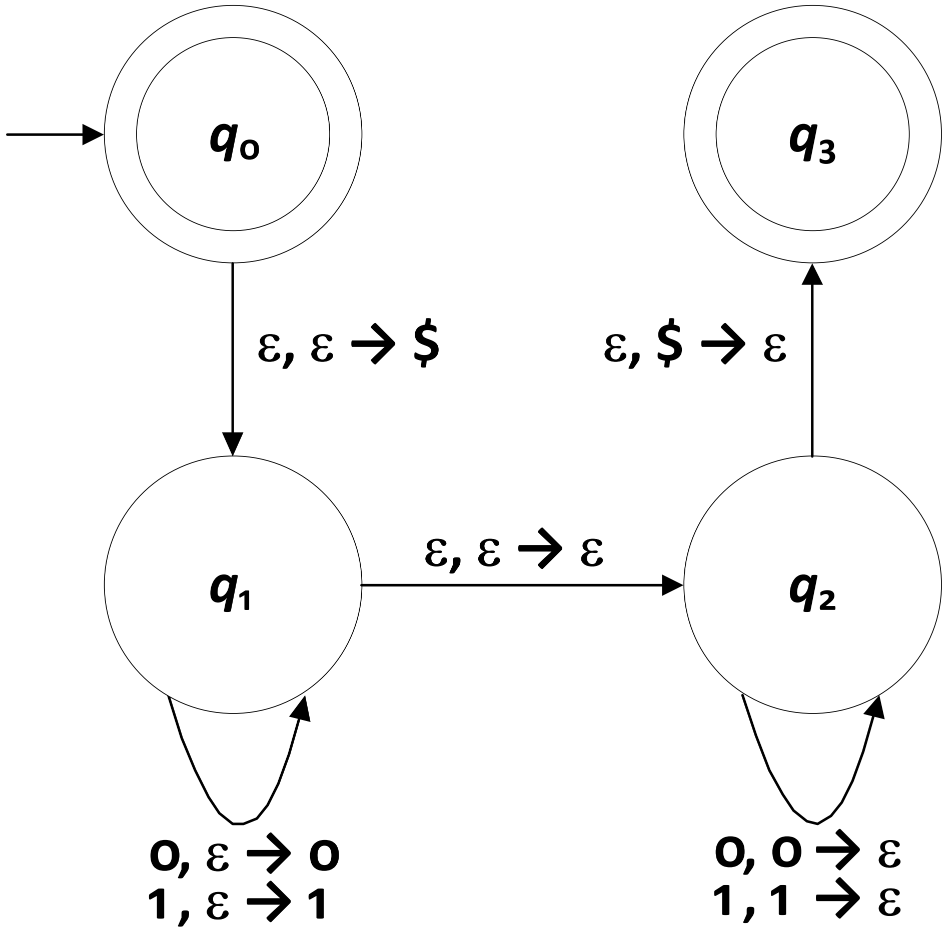
   So any input that arrives causes the computation to die and the string is rejected.



PDA STATE DIAGRAMS

- Case Study 2:

  The PDA P_2 that can recognize the non-regular language $L_2 = \{ww^R \mid w \in \{0, 1\}^*\}$ is:



PDA STATE DIAGRAMS

- The PDA that recognizes language L_2 should perform the following:

1. The PDA reads each symbol of the input string.

2. For each input symbol that it receives, the PDA pushes it onto the stack.

3. When the middle of the string is reached, the PDA starts popping off the stack for each input symbol read, trying to determine if what is read at the input and what is popped off the stack is the same symbol.

PDA STATE DIAGRAMS

Remember, the stack follows the LIFO structure so popping the string inside the stack causes the symbols to come out in the reverse order.

4. If each symbol read from the input and popped out from the stack is always the same and the stack empties at the same time as input is finished, the string is accepted.

Otherwise, it is rejected.

PDA STATE DIAGRAMS

- How will the PDA know if it has reached the middle of the string?

  This is where the nondeterministic nature of the PDA will come into the picture.

  Each time an input symbol is received, the PDA can make a nondeterministic guess that it has reached the middle of the string.

  In other words, upon receiving an input symbol, another copy of the PDA will be created

The new copy will assume that the middle has been reached and will start comparing the input symbol with the symbol at the top of the stack.

The original copy will assume that the middle has not been reached and will continue pushing the input symbols onto the stack.

Eventually, one copy of the PDA will reach the final state if the input string belongs to language L_2.

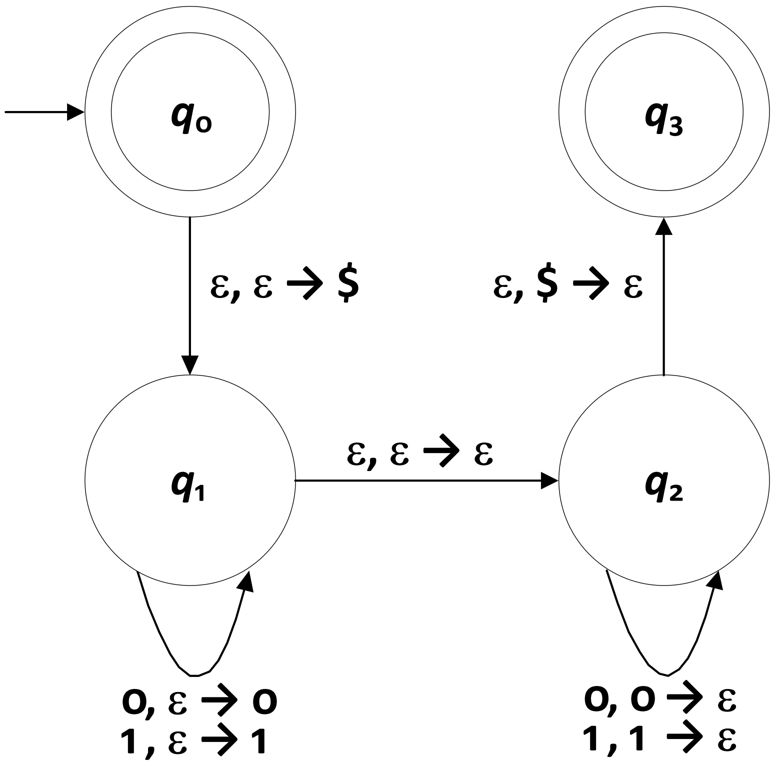
PDA STATE DIAGRAMS

- PDA P_2 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 → $**.

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



PDA STATE DIAGRAMS

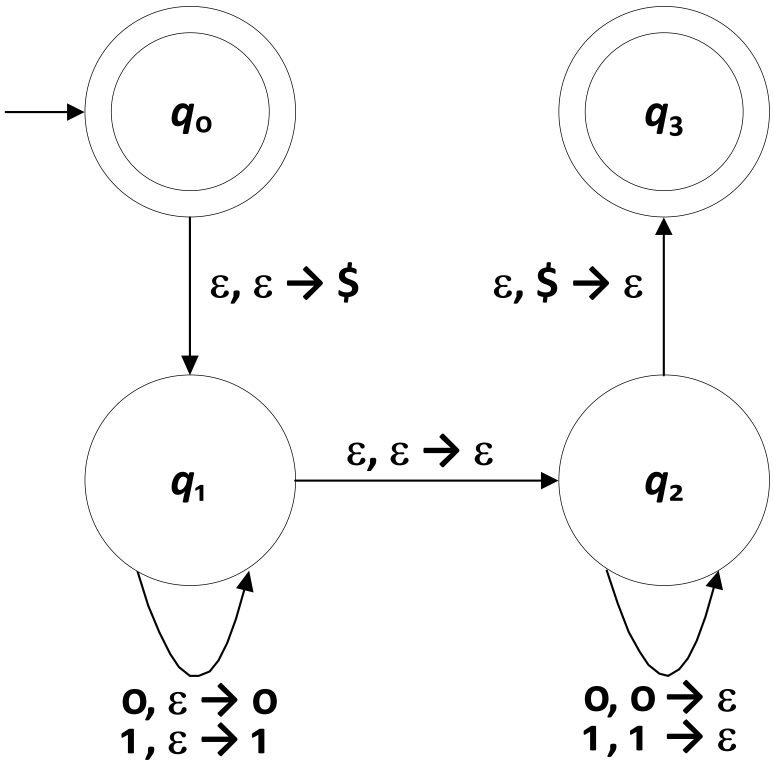
- PDA P_2 Operation

3. At state q_1, there are two transition edges.

   One transition edge has the labels **0, ε → 0** and **1, ε → 1** and goes back to state q_1.

   This is the part where the PDA continuously pushes the incoming input symbols onto the stack.

   In other words, this is where the PDA assumes that it has not yet reached the middle of the string.



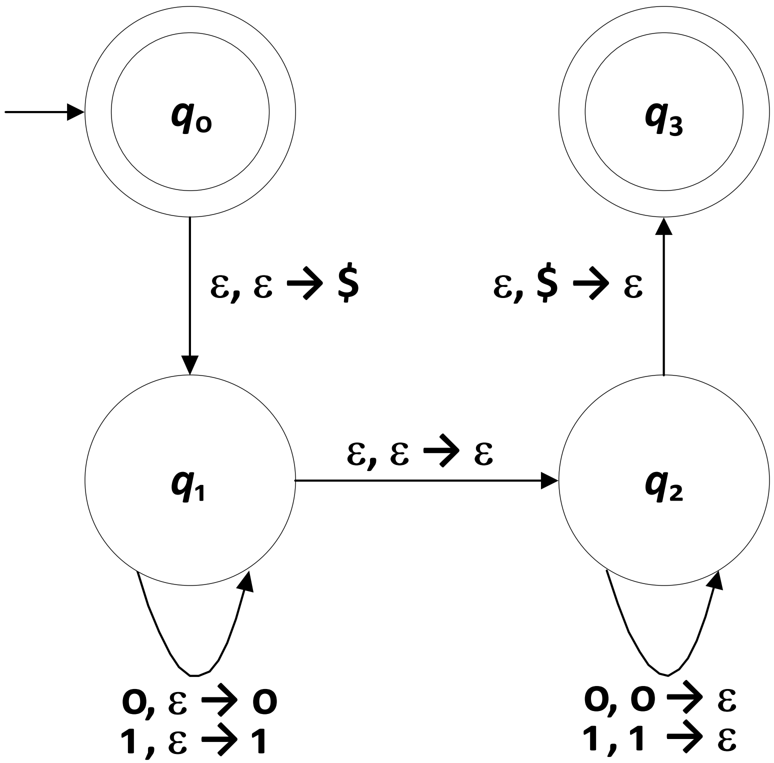
PDA STATE DIAGRAMS

- PDA P_2 Operation

The other transition edge has the label **ε, ε → ε** which leads to state q_2.

This transition edge gives the PDA the option of automatically going to state q_2 without waiting for an input symbol, without checking the symbol at the top of the stack, and without pushing anything onto the stack.

This is the part where the PDA assumes that it has reached the middle of the string.



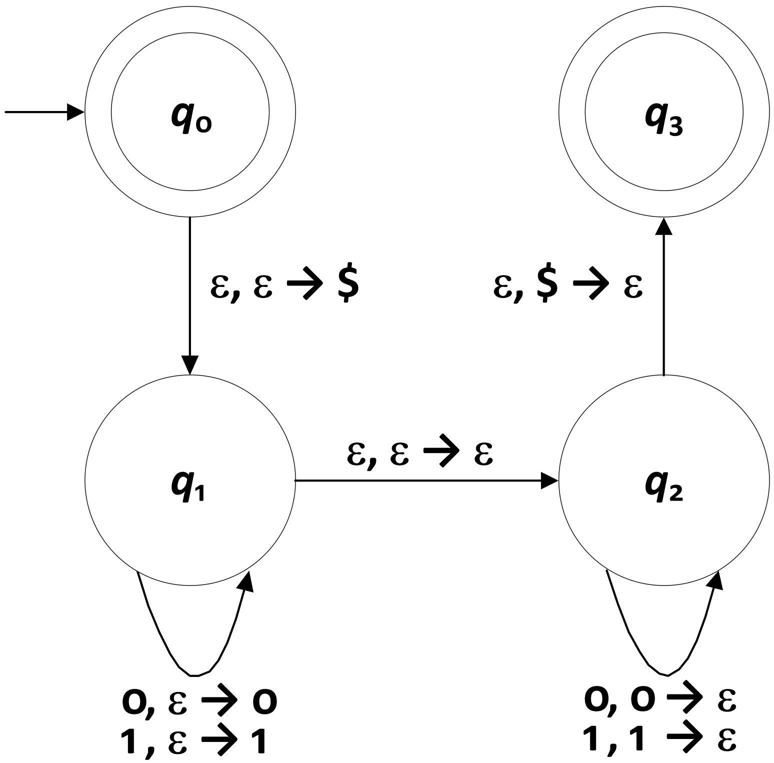
PDA STATE DIAGRAMS

- PDA P_2 Operation

At this point, there will always be two copies of the PDA for each input symbol that arrives.

One copy assumes that the middle of the string has not been reached and will continue pushing the incoming input symbols onto the stack.

The second copy assumes that the middle had been reached and will go to state q_2 where the actual checking will be done.



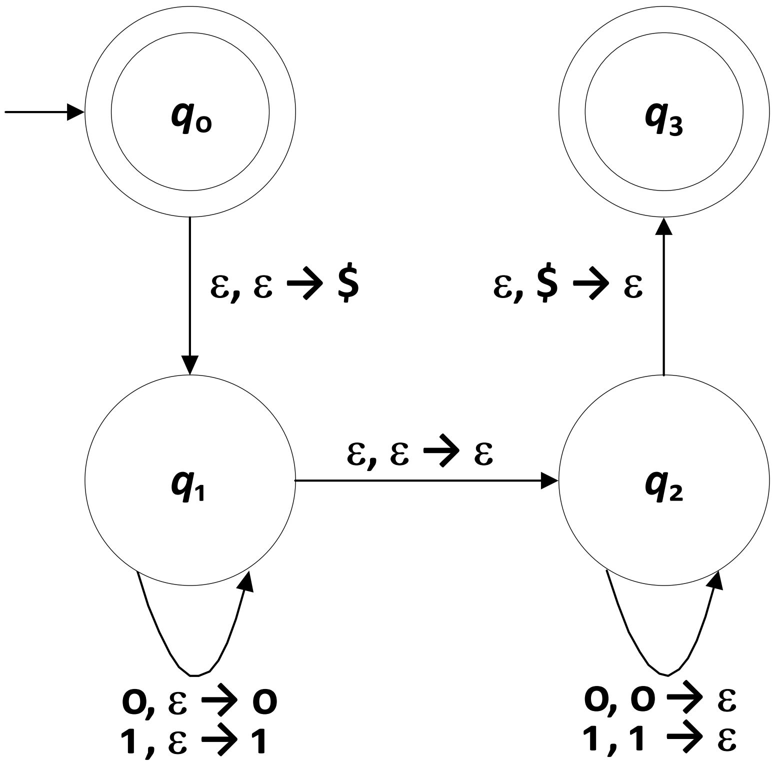
PDA STATE DIAGRAMS

- PDA P_2 Operation

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

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

   As mentioned earlier, this is the point where the PDA starts popping off the symbol at the top of the stack and comparing it with the incoming input symbol

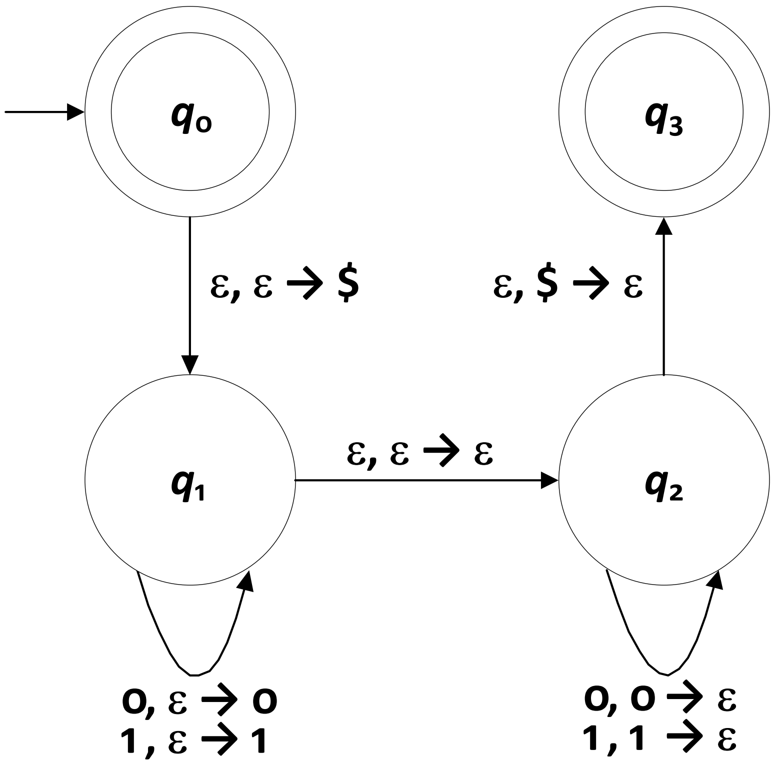


PDA STATE DIAGRAMS

- PDA P_2 Operation

This continues while there is always a match (the PDA keeps going back to state q_2).

If the symbol at the top of the stack and the incoming symbol are not equal, the PDA cannot go anywhere and the computation dies (the string is rejected).



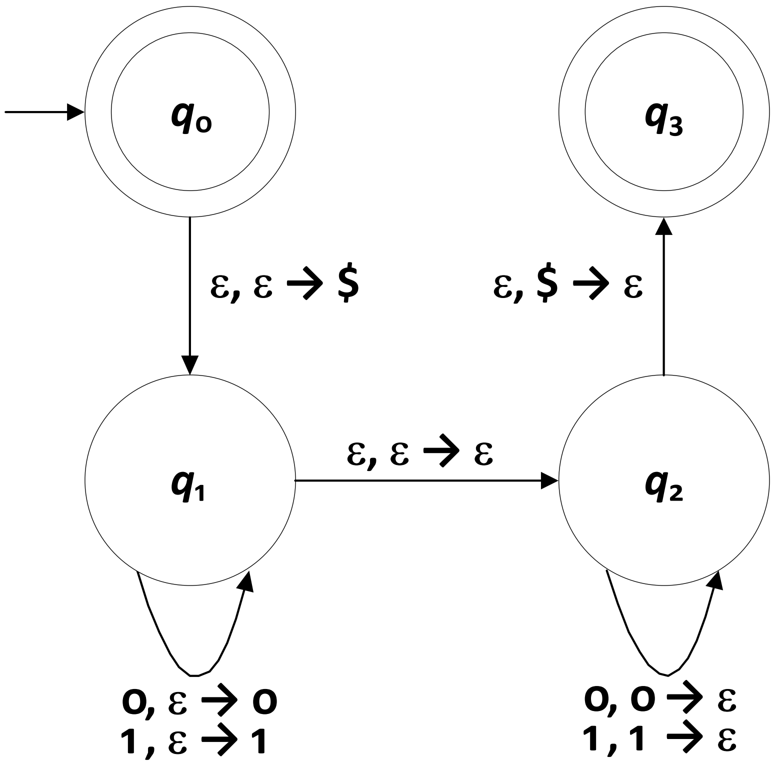
PDA STATE DIAGRAMS

- PDA P_2 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 the second half of the string matches the reverse of the first half.

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



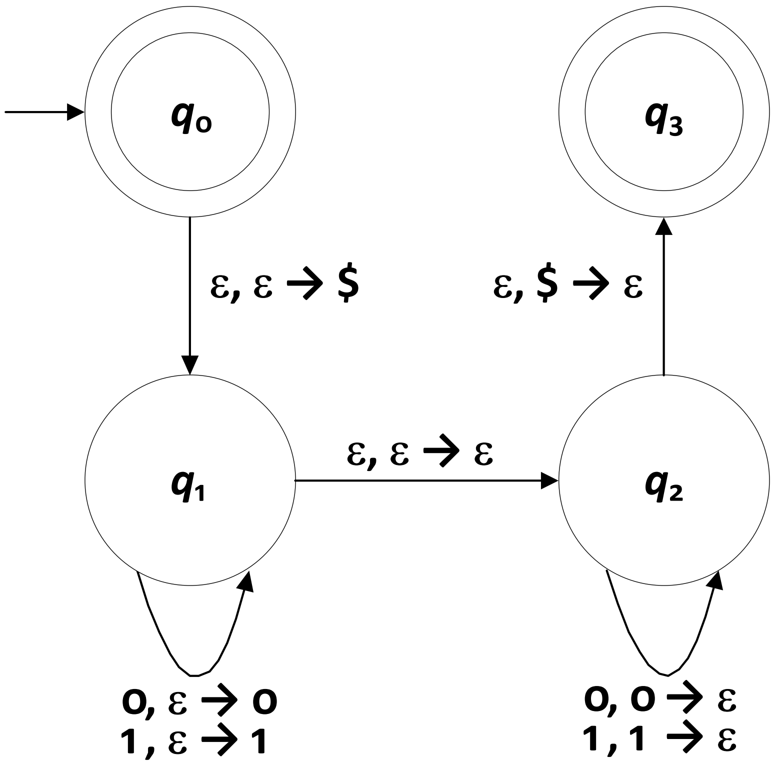
PDA STATE DIAGRAMS

- PDA P_2 Operation

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

   This is where the second half of the string is equal to the reverse of the first half.

   A 0 or a 1 that arrives at this point in time destroys the required pattern so the computation stops and the string is therefore rejected.



PDA DESIGN

- Construct a PDA that can recognize the language $L_4 = \{w \mid w$ is composed of an equal number of 0s and 1s$\}$.

  The language L_4 is the set of all strings that have an equal number of 0s and 1s (not necessarily consecutive 0s and 1s as in language L_1).

  The approach here is to keep track of whether there are more 0s than 1s or whether there are more 1s than 0s.

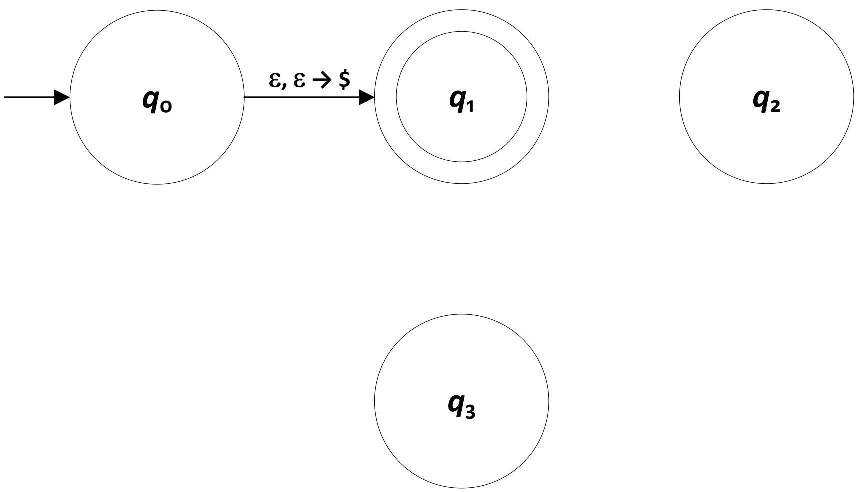
  To do this, the PDA should push any excess symbols (whether 0s or 1s) onto the stack. Hence, if the stack is empty, then there are an equal number of 0s and 1s.

PDA DESIGN

- The PDA should have states for the following situations:

  1. There are an equal number of 0s and 1s. Call this state q_1 and this will be a final state.
  2. There are more 0s than 1s. Call this state q_2.
  3. There are more 1s than 0s. Call this state q_3.

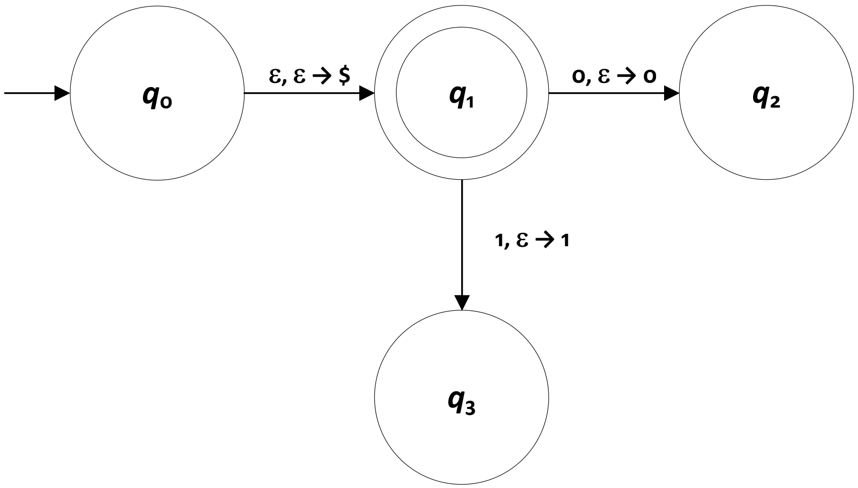
- As usual, there should be a start state q_0 with a transition edge to q_1 with the label **ε, ε → $**. This pushes the *$* symbol onto the stack before any computation could begin.



PDA DESIGN

- While the PDA is in state q_1, a 0 that arrives causes the PDA to push the 0 onto the stack (it's an excess 0) and the PDA goes to state q_2.

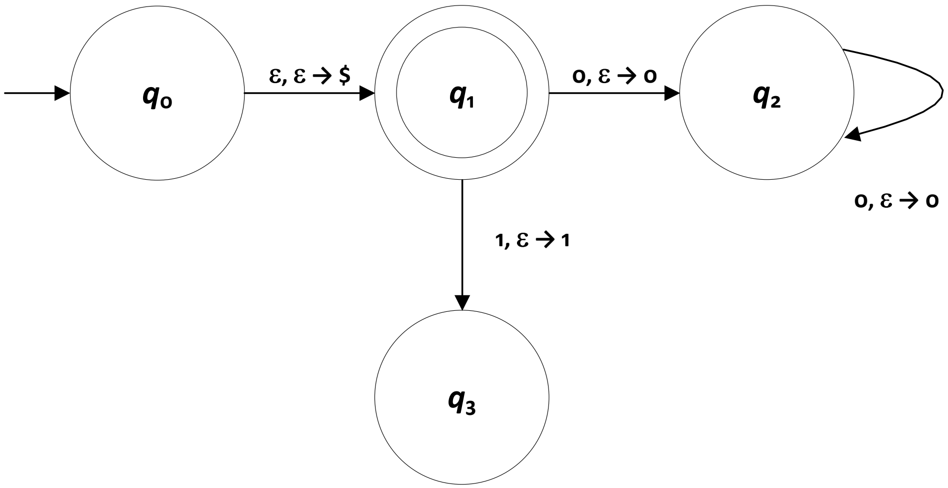
  A 1 that arrives causes the PDA to push the 1 onto the stack (it's an excess 1) and the PDA goes to state q_3.



PDA DESIGN

- While the PDA is in state q_2, any 0 that will arrive will be pushed onto the stack and the PDA stays at q_2.

  Take note that the number of 0s in the stack indicates the number of excess 0s.

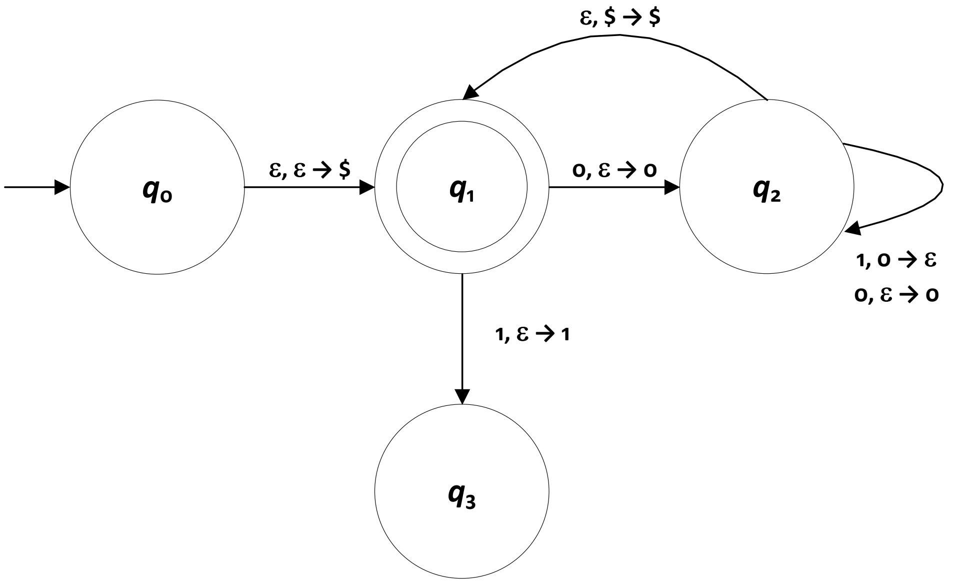


PDA DESIGN

- For any input symbol 1 that arrives, the PDA will pop a 0 from the stack signifying that there is now one less excess 0. The PDA stays at q_2.

- If the PDA sees that the stack is empty (the *$* symbol was popped out), then there are no more excess 0s. The PDA then goes to state q_1.

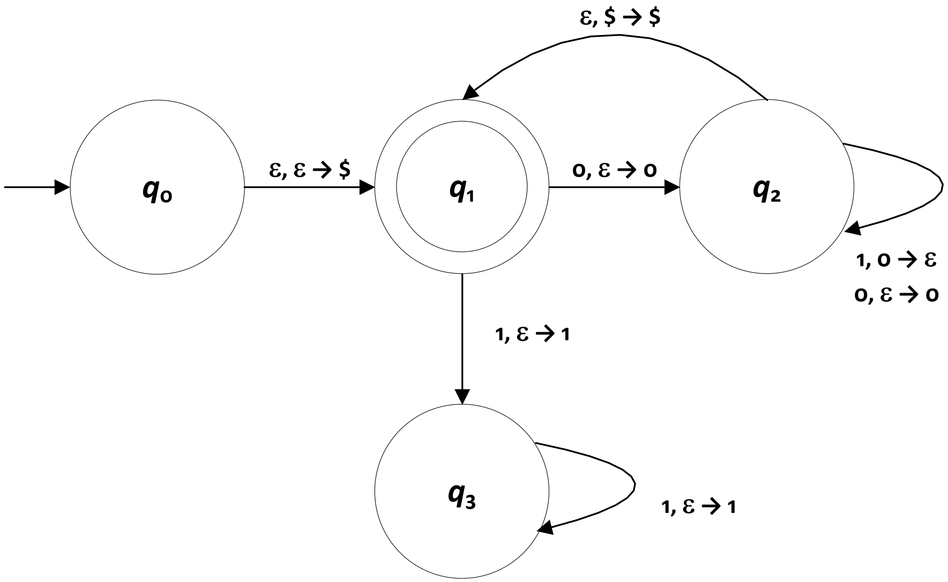
- But because of the **ε, $ → $** transition edge, the PDA goes to q_1 and restores the *$* symbol (since it was popped out earlier).



PDA DESIGN

- While the PDA is in state q_3, any 1 that will arrive will be pushed onto the stack.

  The number of 1s in the stack indicates the number of excess 1s.

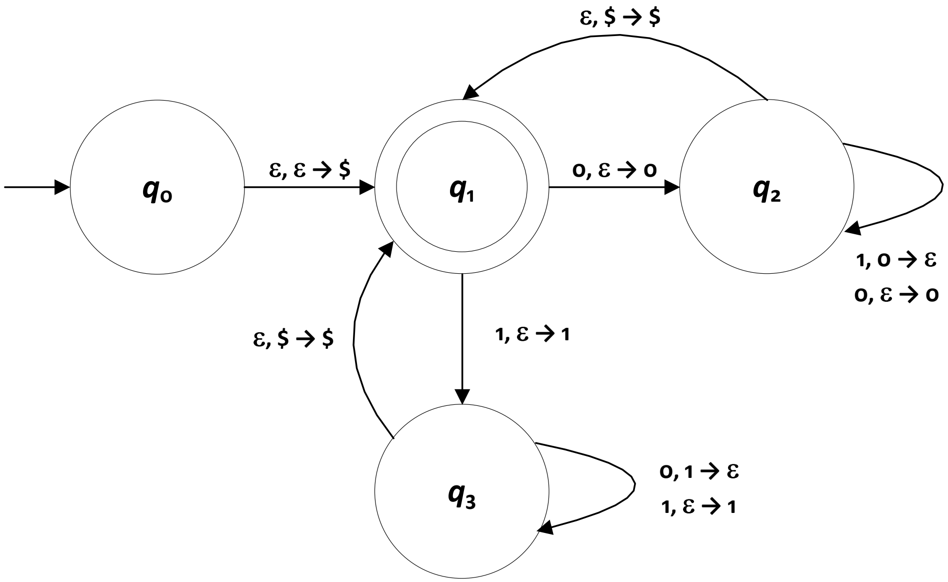


PDA DESIGN

- For any input symbol 0 that arrives, the PDA will pop a 1 from the stack signifying that there is now one less excess 1.

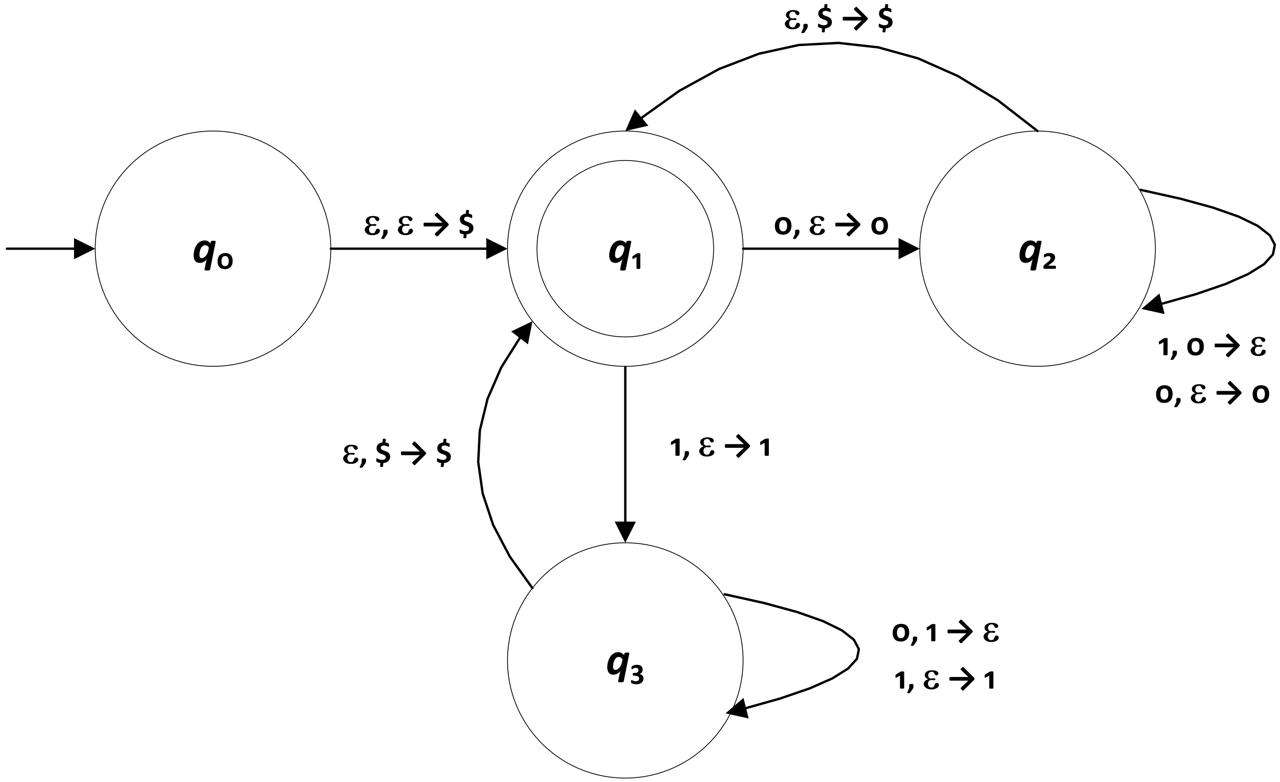
- If the PDA sees that the stack is empty (the $ symbol was popped out), then there are no more excess 1s.

- The PDA then goes to state q_1 and pushes the *$* symbol onto the stack to restore it.



PDA STATE DIAGRAMS

- So the PDA that can recognize the language L_4 = {w| w is composed of an equal number of 0s and 1s} is:



PDA DESIGN

- Construct a PDA that can recognize the language $L_5 = \{0^{2n}1^n \mid n \geq 1\}$.

  This language is composed of all strings wherein the total number of consecutive 0s is equal to the total number of consecutive 1s following it.

  The language L_5 is composed of all strings wherein the total number of consecutive 0s is two times the total number of consecutive 1s following it. The approach here is similar to the PDA P_1 presented earlier.

  1. For each input symbol 0 that it receives, the PDA pushes it onto the stack. The stack therefore keeps track of the number of consecutive 0s that the PDA receives.

PDA DESIGN

2. Now for each input symbol 1 that it receives, the PDA pops or removes two 0s from the stack.

3. If there are two 0s that can be popped from the stack for every 1 the PDA receives, then each 1 has two 0s as its "partner."

   Hence, if the number of consecutive 0s is two times the number of consecutive 1s following the 0s, the stack will be empty by the time the input string ends.

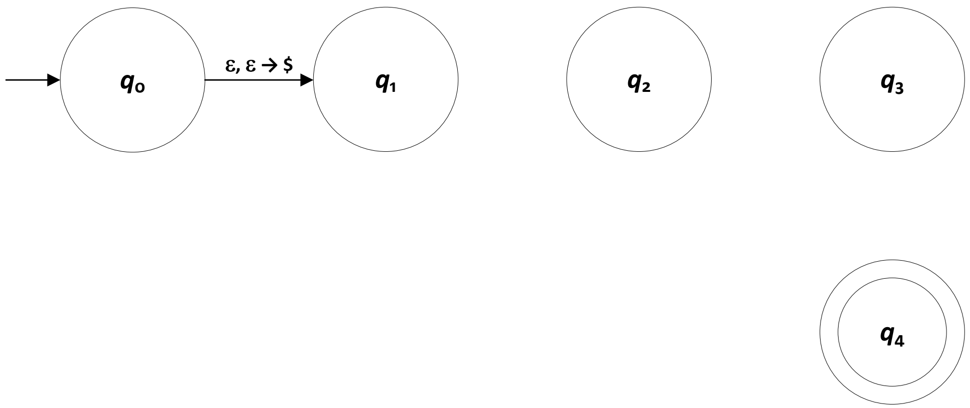
4. If there is no more input symbol and the stack is empty, the input string is accepted.

PDA DESIGN

- The PDA should have states for the following situations:

  1. The arrival of 0s and each 0 is pushed onto the stack. Call this state q_1.
  2. The arrival of a 1 causing a 0 to be popped from the stack. Call this state q_2.
  3. The popping of another 0 for the 1 that arrived in state q_2. Call this state q_3.
  4. The state when there are no more input symbols and the stack is empty. This is the final state and it will be called q_4.

- As usual, there should be a start state q_0 with a transition edge to q_1 with the label **ε, ε → $**. This pushes the $ symbol onto the stack before any computation could begin.

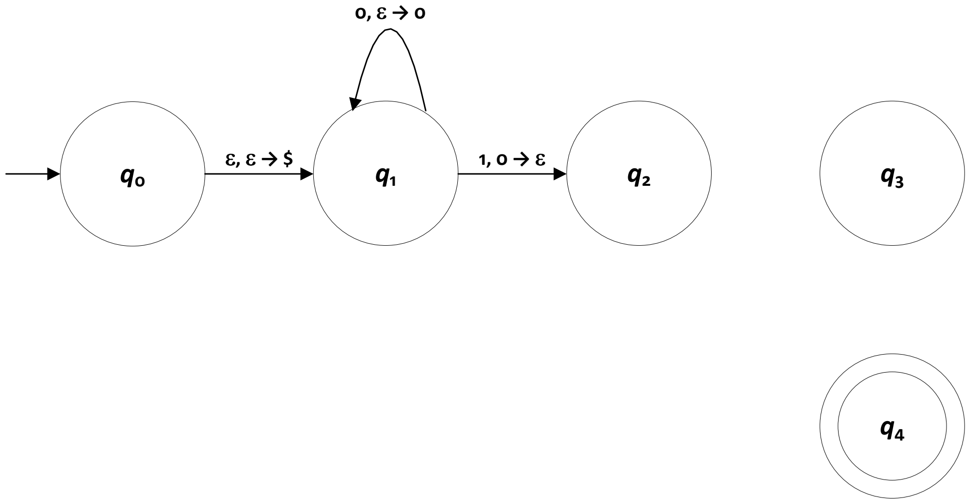


PDA DESIGN

- While the PDA is in state q_1, a 0 that arrives causes the PDA to push the 0 onto the stack and the PDA stays at state q_1.

  A 1 that arrives signals the start of the arrival of consecutive 1s.

  A 0 will now be popped as one of the two 0s which will be partnered with the 1 that just arrived. The PDA goes to state q_2.

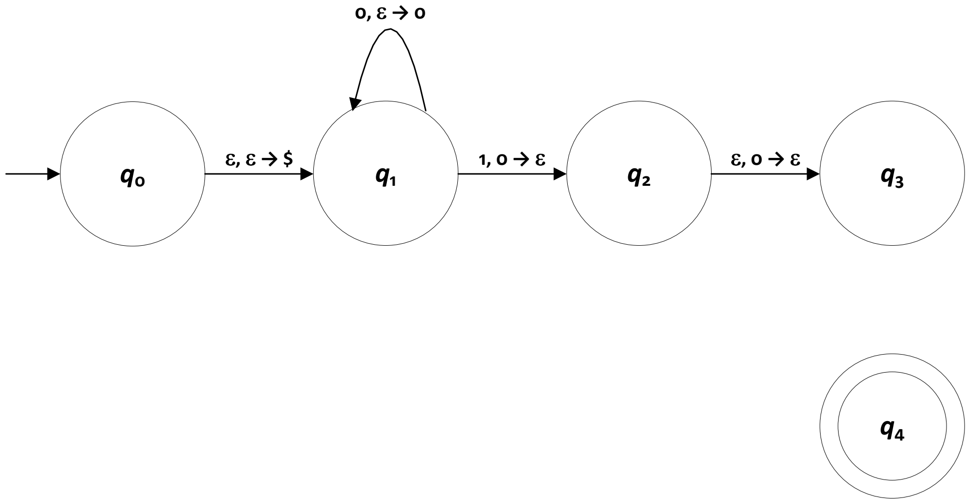


PDA DESIGN

- At state q_2, the PDA will automatically pop another 0 (assuming there are still 0s inside the stack) as the 2nd partner of the 1 that arrived while at state q_1.

  This will happen without waiting for a new input symbol to arrived.

  If there are no 0s to be popped (the stack is empty) while the PDA is at q_2, then the computation stops and the string is rejected.



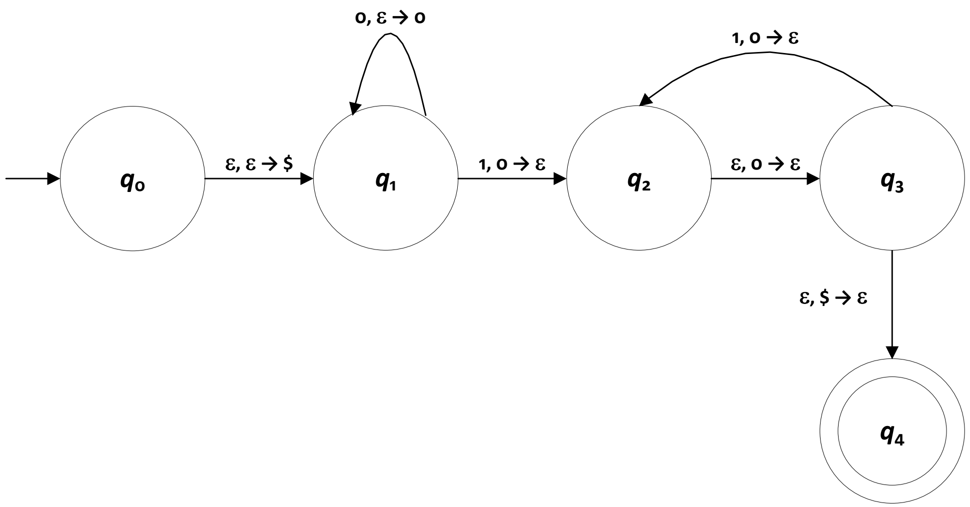
PDA DESIGN

- While at state q_3, if a 1 arrives, a 0 will again be popped as one of the two 0s which will be partnered with the 1 that just arrived. The PDA goes to state q_2.

- If there are no more 0s to be popped, the computation stops.

  A 0 that arrives at this point in an out-of-sequence 0 so the computation dies.

  If there are no more input symbols and the stack is empty, then the PDA goes to the final state q_4 and the string is accepted.



EXERCISES

- So the PDA that can recognize the language $L_5 = \{0^{2n}1^n \mid n \geq 1\}$: