THEORY OF COMPUTATIONS

BCA 5th Semester

Formal Languages

- In mathematics, computer science, and linguistics, a **formal language** consists of words whose letters are taken from an alphabet and are well-formed according to a specific set of rules.

Formal Languages

- The alphabet of a formal language consist of symbols, letters, or tokens that concatenate into strings of the language.[1] Each string concatenated from symbols of this alphabet is called a word, and the words that belong to a particular formal language are sometimes called *well-formed words* or *well-formed formulas*. A formal language is often defined by means of a formal grammar such as a regular grammar or context-free grammar, which consists of its formation rules.

Alphabets

- Theory of computation is entirely based on symbols. These symbols are generally letters and digits.
  Alphabets are defined as *a finite set of symbols.*
  Examples:
  $\sum = \{0, 1\}$ is an alphabet of binary digits
  $\sum = \{A, B, C, \ldots., Z\}$ is an alphabet.

Strings

- A string is *a finite sequence of symbols selected from some alphabet*. It is generally denoted as w. For example for alphabet ∑ = {0, 1} w = 010101 is a string.

- *Length of a string* is denoted as |w| and is defined as the number of positions for the symbol in the string. For the above example length is 6.

Strings

- The *empty string* is the string with zero occurrence of symbols. This string is represented as ϵ or λ.
- The set of strings, including the empty string, over an alphabet ∑ is denoted by ∑*.
  For ∑ = {0, 1} we have set of strings as ∑* = {ϵ, 0, 1, 01, 10, 00, 11, 10101,...}.
  and ∑1 = {0, 1} , ∑2 = {00, 01, 10, 11} and so on.
- ∑* contains an empty string ϵ. The set of non-empty string is denoted by ∑+. From this we get:
  ∑* = ∑+ U {ϵ }

Concatenation of strings

- Let w1 and w2 be two strings then w1w2 denotes their concatenation w. The concatenation is formed by making a copy of w1 and followed by a copy of w2.
  For example w1 = xyz, w2 = uvw
  then w = w1w2 = xyzuvw
  Also w13 = w1w1w1

Languages

- A language is a *set of string all of which are chosen from some ∑*, where ∑ is a particular alphabet*. This means that language L is subset of ∑*. An example is English language, where the collection of legal English words is a set of strings over the alphabet that consists of all the letters. Another example is the C programming language where the alphabet is a subset of the ASCII characters and programs are subset of strings that can be formed from this alphabet.

Concatenation of Languages

- If L1 and L2 are two languages then their concatenation can be defined as :
  L = L1 . L2 where L = {w: w = xy where x є L1, y є L2}
  It means that all the strings in the language L are concatenation of stings of language L1 and L2

Kleen Closure

- If S is a set of words then by S* we mean the set of all finite strings formed by concatenating words from S, where any word may be used as often we like, and where the null string is also included.
- S* is the Kleen closure for S. We can think of kleen star (S*) as an operation that makes an infinite language of strings of letters out of an alphabet.
  For example for ∑ = {a}
  ∑* = {є, a, aa, aaa, ….}

Automata

- An automaton (Automata in plural) is an abstract self-propelled computing device which follows a predetermined sequence of operations automatically.

- An automaton with a finite number of states is called a **Finite Automaton**(FA) or **Finite State Machine** (FSM).

Finite Automaton - Formal definition

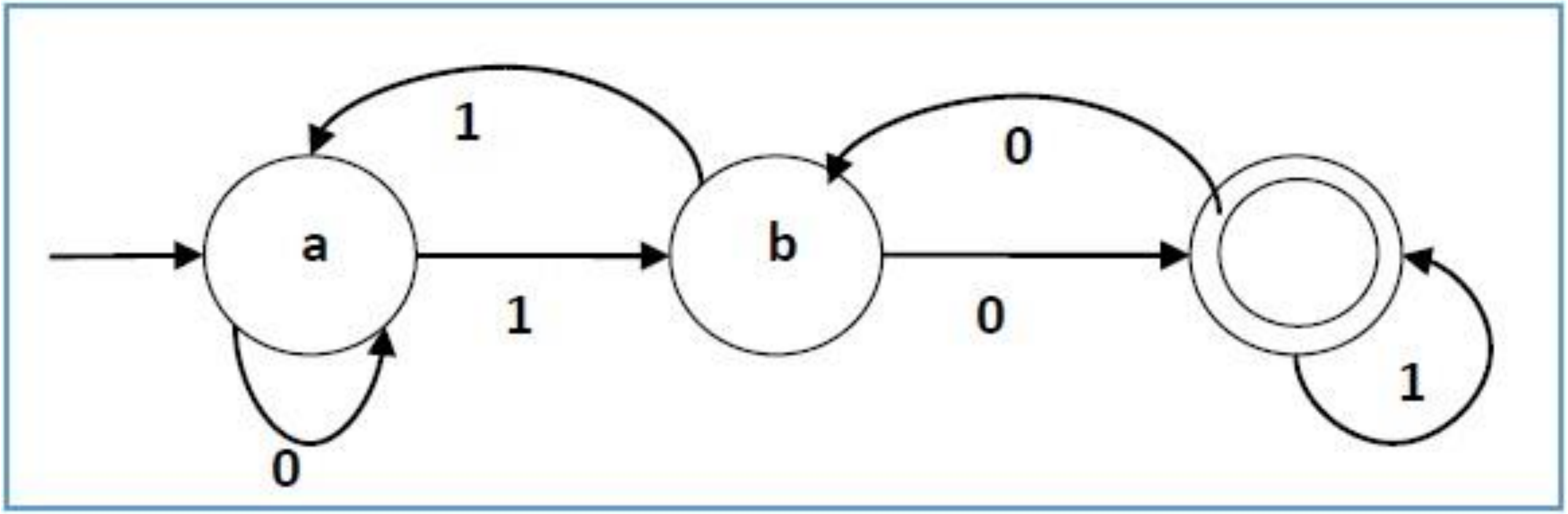
- An automaton can be represented by a quintuple (5-tuple) $(Q, \sum, \delta, q_0, F)$, where –
  - **Q** is a finite set of states.
  - **∑** is a finite set of symbols, called the **alphabet** of the automaton.
  - **δ** is the transition function.
  - $\mathbf{q_0}$ is the initial state from where any input is processed ($q_0 \in Q$).
  - **F** is a set of final state/states of Q ($F \subseteq Q$).

Graphical Representation of a DFA

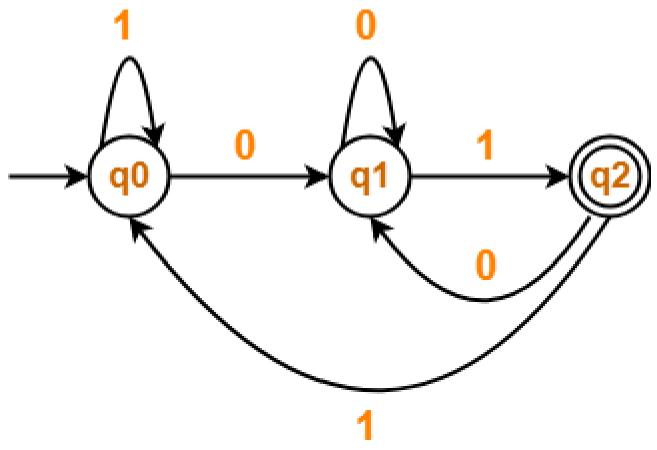
- A DFA is represented by digraphs called **state diagram**.
  - The vertices represent the states.
  - The arcs labelled with an input alphabet show the transitions.
  - The initial state is denoted by an empty single incoming arc.
  - The final state is indicated by double circles.

- Let a deterministic finite automaton be →
- Q = {a, b, c},
- ∑ = {0, 1},
- q_0 = {a},
- F = {c}, and
- Transition function δ as shown by the following table −

Present State	Next State for Input 0	Next State for Input 1
a	a	b
b	c	a
c	b	c

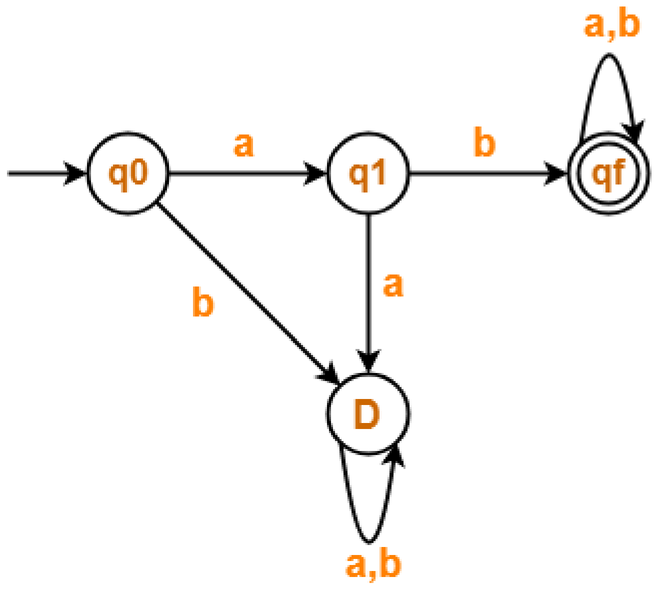

1
0
a
b
1
0
1
0

- Draw a DFA for the language accepting strings ending with ’01’ over input alphabets ∑ = {0, 1}



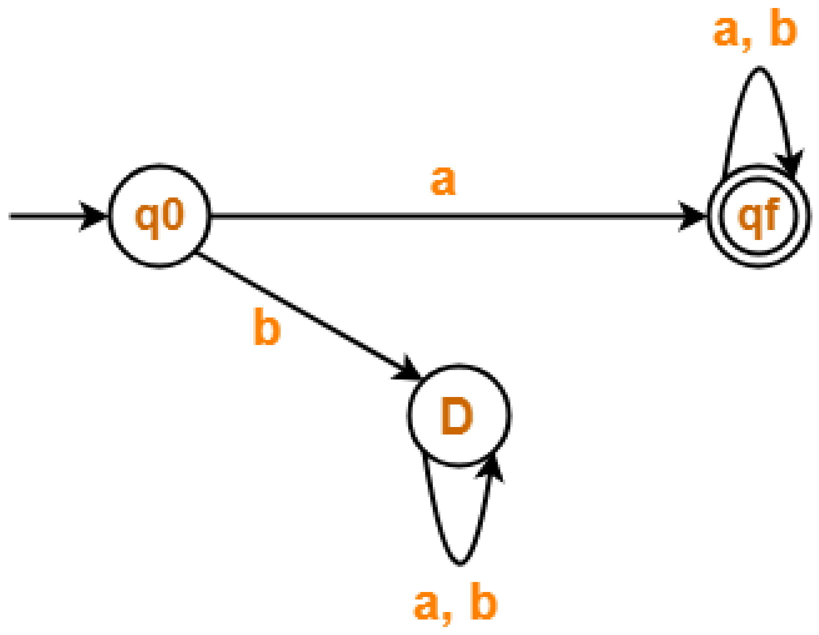
DFA

- Draw a DFA for the language accepting strings starting with ‘ab’ over input alphabets ∑ = {a, b}



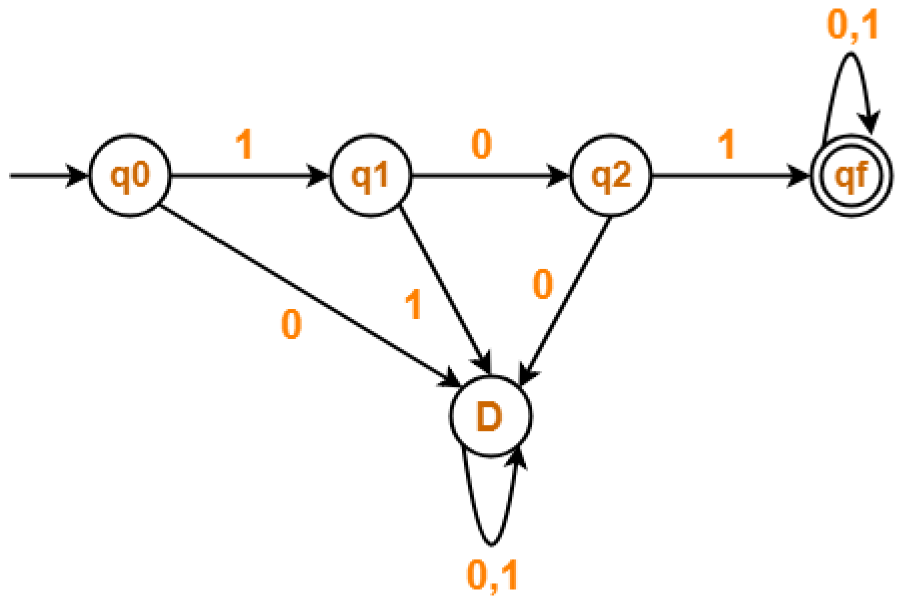
DFA

- Draw a DFA for the language accepting strings starting with ‘a’ over input alphabets ∑ = {a, b}



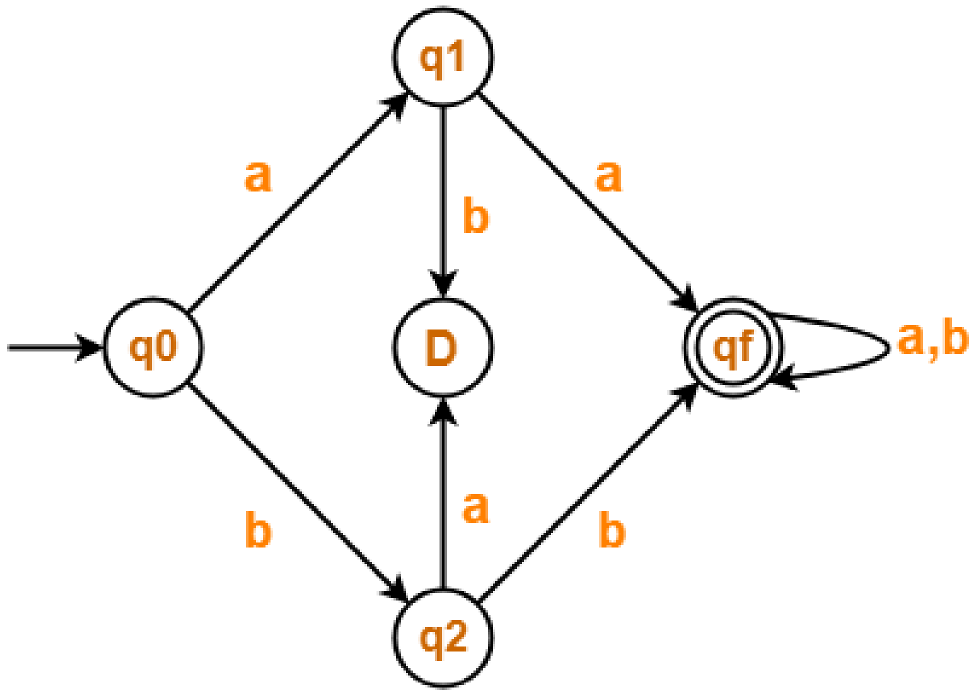
DFA

- Draw a DFA for the language accepting strings starting with ‘101’ over input alphabets ∑ = {0, 1}



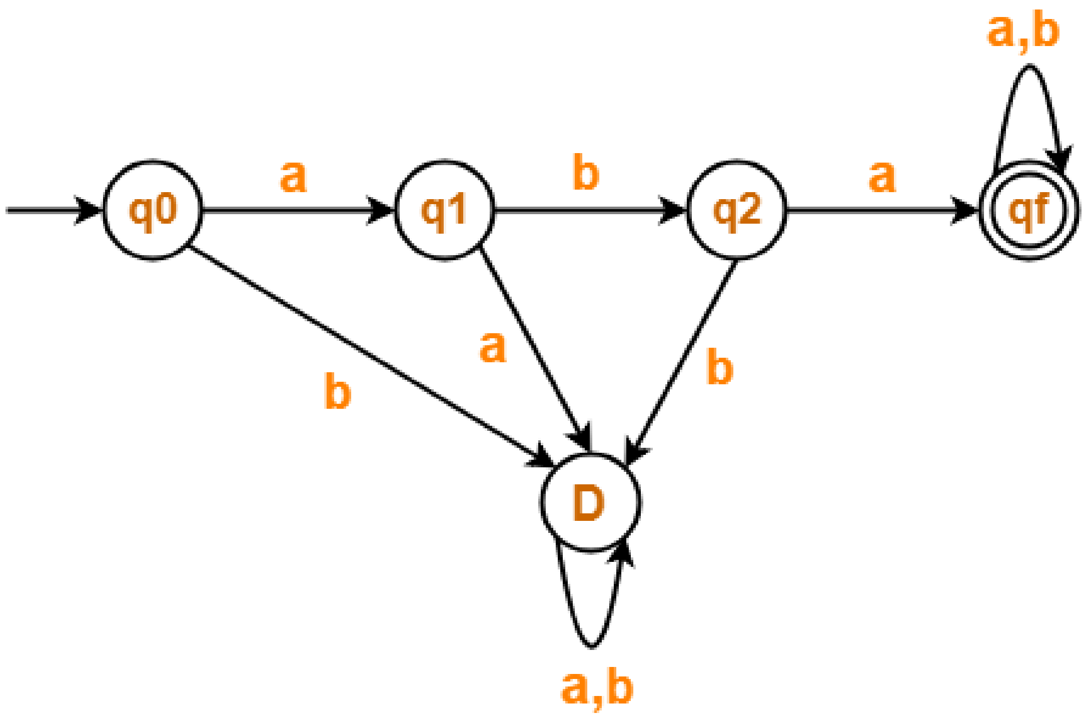
DFA

- Construct a DFA that accepts a language L over input alphabets ∑ = {a, b} such that L is the set of all strings starting with ‘aa’ or ‘bb’.



DFA

- Construct a DFA that accepts a language L over input alphabets ∑ = {a, b} such that L is the set of all strings starting with 'aba'.


$$\begin{tikzcd}
\arrow[r] & q0 \arrow[r, "a"] \arrow[dr, "b"'] & q1 \arrow[r, "b"] \arrow[d, "a"] & q2 \arrow[r, "a"] \arrow[dl, "b"] & qf \arrow[loop above, "{a,b}"] \\
 & & D \arrow[loop below, "{a,b}"] & &
\end{tikzcd}$$

DFA

Non-deterministic Automaton

- In NDFA, for a particular input symbol, the machine can move to any combination of the states in the machine. In other words, the exact state to which the machine moves cannot be determined. Hence, it is called **Non-deterministic Automaton**. As it has finite number of states, the machine is called **Non-deterministic Finite Machine** or **Non-deterministic Finite Automaton**.

Formal Definition of an NDFA

- An NDFA can be represented by a 5-tuple (Q, ∑, δ, q_0, F) where –
- **Q** is a finite set of states.
- **∑** is a finite set of symbols called the alphabets.
- **δ** is the transition function where $\delta: Q \times \sum \rightarrow 2^Q$
- (Here the power set of Q (2^Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)
- **q_0** is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

Graphical Representation of an NDFA

- An NDFA is represented by digraphs called state diagram.
  - The vertices represent the states.
  - The arcs labelled with an input alphabet show the transitions.
  - The initial state is denoted by an empty single incoming arc.
  - The final state is indicated by double circles.

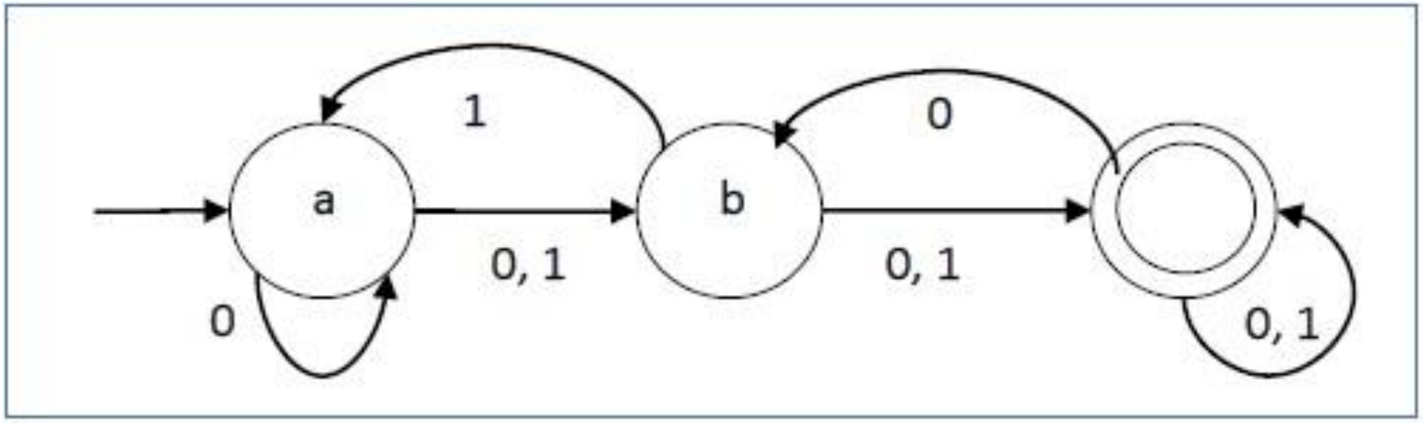
Example

- Let a non-deterministic finite automaton be →
- Q = {a, b, c}
- ∑ = {0, 1}
- q_0 = {a}
- F = {c}

Example

The transition function δ as shown below −

Present State	Next State for Input 0	Next State for Input 1
a	a, b	b
b	c	a, c
c	b, c	c



Regular Expressions

- Just as finite automata are used to *recognize* patterns of strings, regular expressions are used to *generate* patterns of strings.

- A regular expression is an algebraic formula whose value is a pattern consisting of a set of strings, called the language of the expression.

Regular Expressions

- Operands in a regular expression can be:
  - *characters* from the alphabet over which the regular expression is defined.
  - *variables* whose values are any pattern defined by a regular expression.
  - *Epsilon* **(ε)** which denotes the empty string containing no characters.
  - *null* which denotes the empty set of strings.

Regular Expressions

- Operators used in regular expressions include:
  - Union: If R1 and R2 are regular expressions, then R1 | R2 (also written as R1 U R2 or R1 + R2) is also a regular expression. L(R1|R2) = L(R1) U L(R2).
  - Concatenation: If R1 and R2 are regular expressions, then R1R2 (also written as R1.R2) is also a regular expression. L(R1R2) = L(R1) concatenated with L(R2).

Regular Expressions

- Operators used in regular expressions include:
  - Kleene closure: If R1 is a regular expression, then R1* (the Kleene closure of R1) is also a regular expression. L(R1*) = epsilon U L(R1) U L(R1R1) U L(R1R1R1) U ...

Examples

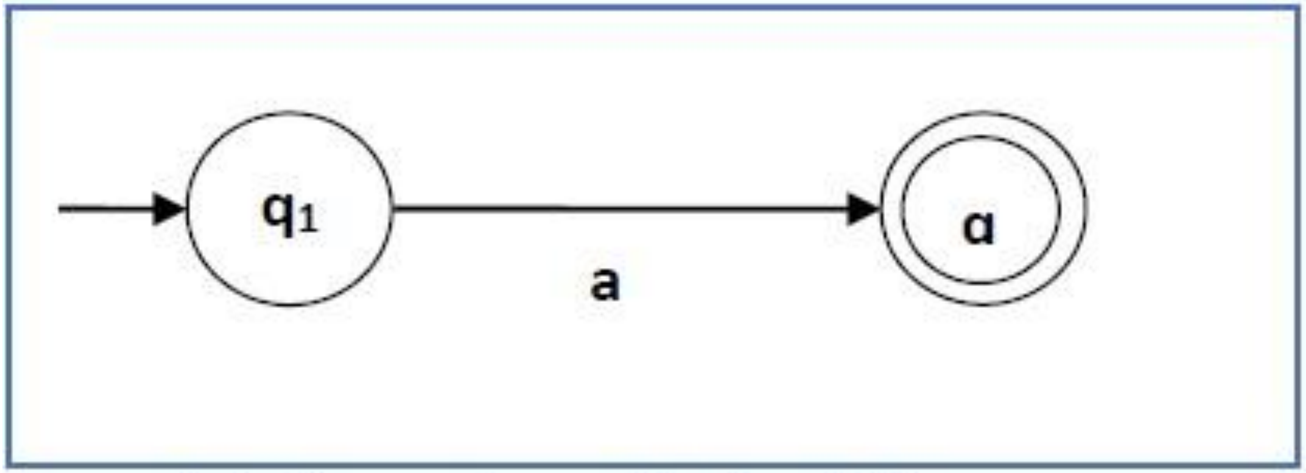
- The set of strings over {0,1} that end in 3 consecutive 1's.
  - (0 | 1)* 111
- The set of strings over {0,1} that have at least one 1.
  - 0* 1 (0 | 1)*
- The set of strings over {A..Z,a..z} that contain the word "main".
  - Let <letter> = A | B | ... | Z | a | b | ... | z <letter>* main <letter>*

Equivalence of Regular Expressions and Finite Automata

- Regular expressions and finite automata have equivalent expressive power:
  - For every regular expression R, there is a corresponding FA that accepts the set of strings generated by R.
  - For every FA A there is a corresponding regular expression that generates the set of strings accepted by A.

Constructing FA from RE

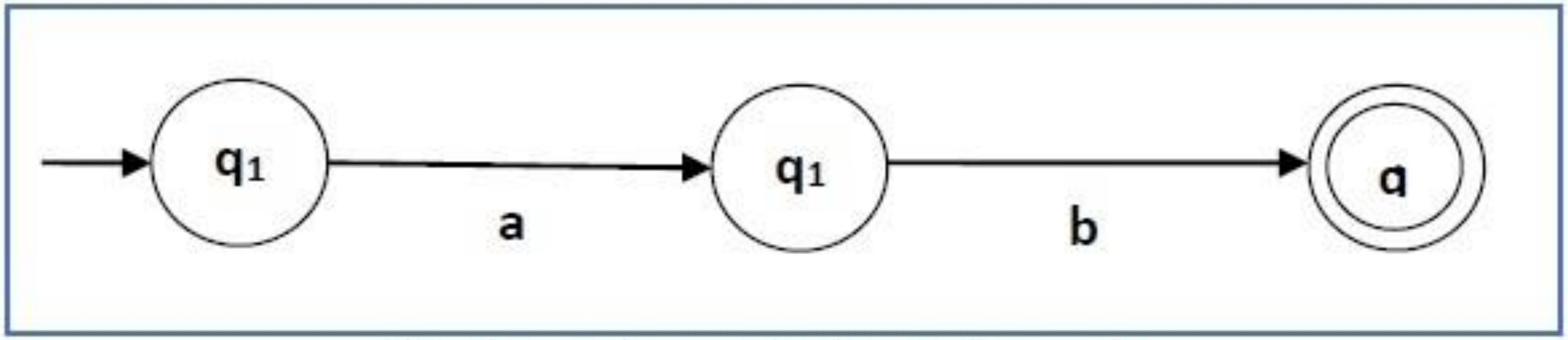
- ***Case 1*** – For a regular expression ‘a’, we can construct the following FA –



Finite automata for RE = a

Constructing FA from RE

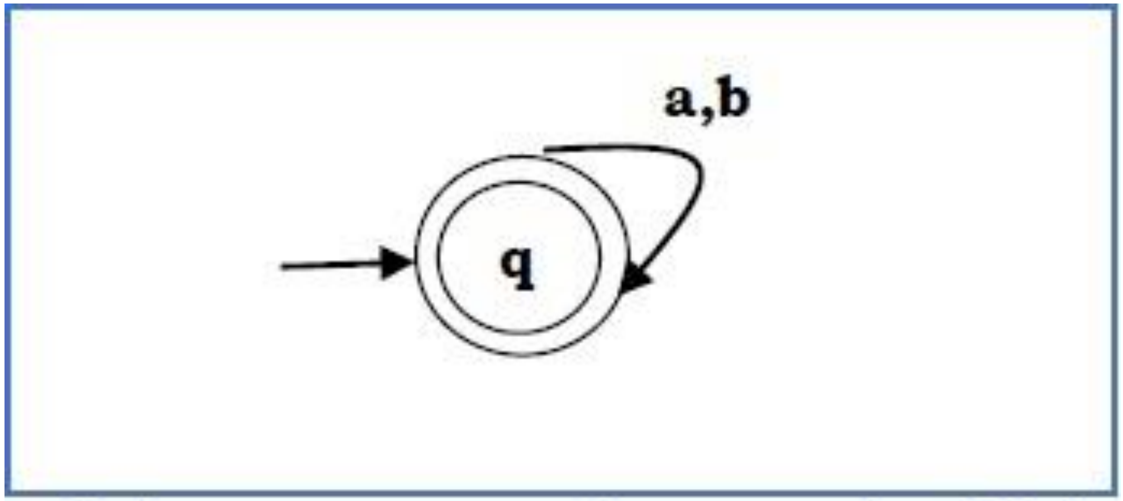
- ***Case 2*** – For a regular expression ‘ab’, we can construct the following FA –



Finite automata for RE = ab

Constructing FA from RE

- ***Case 3*** – For a regular expression (a+b)*, we can construct the following FA –



Finite automata for RE= (a+b)*

Regular Languages

- A **regular language** is a language that can be expressed with a regular expression or a deterministic or non-deterministic finite automata or state machine. A **language** is a set of strings which are made up of characters from a specified alphabet, or set of symbols.

Regular Languages

- Regular languages are a subset of the set of all strings. Regular languages are used in parsing and designing programming languages and are one of the first concepts taught in computability courses.

Pushdown Automata

- A pushdown automaton is a way to implement a context-free grammar in a similar way we design DFA for a regular grammar. A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information.

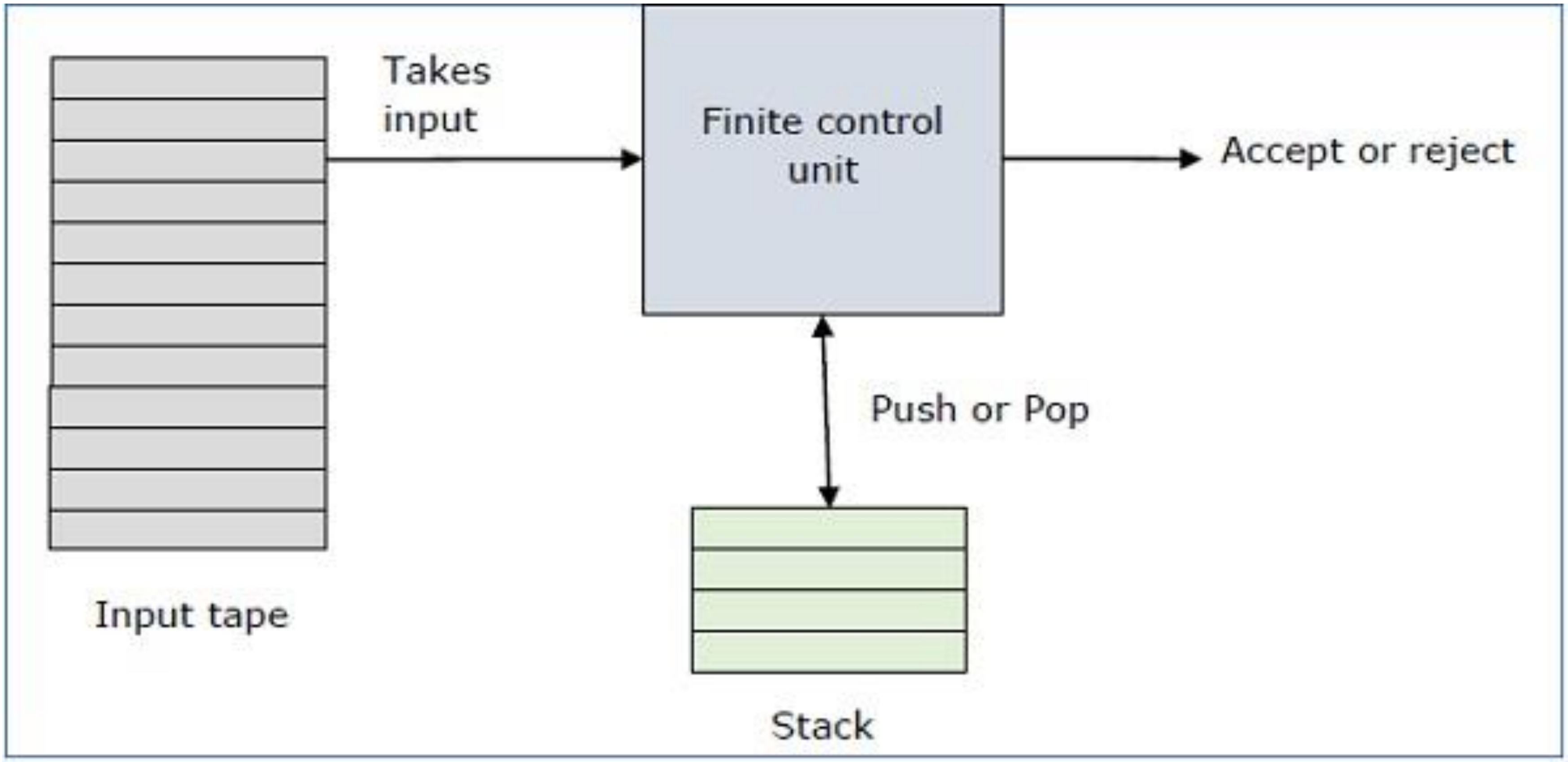
- Basically a pushdown automaton is –

  "Finite state machine" + "a stack"

Pushdown Automata

- A pushdown automaton has three components –
  - an input tape,
  - a control unit, and
  - a stack with infinite size.
- The stack head scans the top symbol of the stack.
- A stack does two operations –
  - **Push** – a new symbol is added at the top.
  - **Pop** – the top symbol is read and removed.

Pushdown Automata



Pushdown Automata

- A PDA can be formally described as a 7-tuple (Q, ∑, S, δ, q_0, I, F) –
  - **Q** is the finite number of states
  - **∑** is input alphabet
  - **S** is stack symbols
  - **δ** is the transition function: Q × (∑ ∪ {ε}) × S × Q × S*
  - **q_0** is the initial state (q_0 ∈ Q)
  - **I** is the initial stack top symbol (I ∈ S)
  - **F** is a set of accepting states (F ∈ Q)

δ: The *transition function.* As for a finite automaton, δ governs the behavior of the automaton. Formally, δ takes as argument a triple $\delta(q, a, X)$, where:

1. q is a state in Q.
2. a is either an input symbol in Σ or $a = \epsilon$, the empty string, which is assumed not to be an input symbol.
3. X is a stack symbol, that is, a member of Γ.

The output of δ is a finite set of pairs (p, γ), where p is the new state, and γ is the string of stack symbols that replaces X at the top of the stack. For instance, if $\gamma = \epsilon$, then the stack is popped, if $\gamma = X$, then the stack is unchanged, and if $\gamma = YZ$, then X is replaced by Z, and Y is pushed onto the stack.

Example 6.2: Let us design a PDA P to accept the language L_{wwr} of Example 6.1. First, there are a few details not present in that example that we need to understand in order to manage the stack properly. We shall use a stack symbol Z_0 to mark the bottom of the stack. We need to have this symbol present so that, after we pop w off the stack and realize that we have seen ww^R on the input, we still have something on the stack to permit us to make a transition to the accepting state, q_2. Thus, our PDA for L_{wwr} can be described as

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

where δ is defined by the following rules:

1. $\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$ and $\delta(q_0, 1, Z_0) = \{(q_0, 1Z_0)\}$. One of these rules applies initially, when we are in state q_0 and we see the start symbol Z_0 at the top of the stack. We read the first input, and push it onto the stack, leaving Z_0 below to mark the bottom.

2. $\delta(q_0, 0, 0) = \{(q_0, 00)\}$, $\delta(q_0, 0, 1) = \{(q_0, 01)\}$, $\delta(q_0, 1, 0) = \{(q_0, 10)\}$, and $\delta(q_0, 1, 1) = \{(q_0, 11)\}$. These four, similar rules allow us to stay in state q_0 and read inputs, pushing each onto the top of the stack and leaving the previous top stack symbol alone.

3. $\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\}$, $\delta(q_0, \epsilon, 0) = \{(q_1, 0)\}$, and $\delta(q_0, \epsilon, 1) = \{(q_1, 1)\}$. These three rules allow P to go from state q_0 to state q_1 spontaneously (on ϵ input), leaving intact whatever symbol is at the top of the stack.

4. $\delta(q_1, 0, 0) = \{(q_1, \epsilon)\}$, and $\delta(q_1, 1, 1) = \{(q_1, \epsilon)\}$. Now, in state q_1 we can match input symbols against the top symbols on the stack, and pop when the symbols match.

5. $\delta(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}$. Finally, if we expose the bottom-of-stack marker Z_0 and we are in state q_1, then we have found an input of the form ww^R. We go to state q_2 and accept.

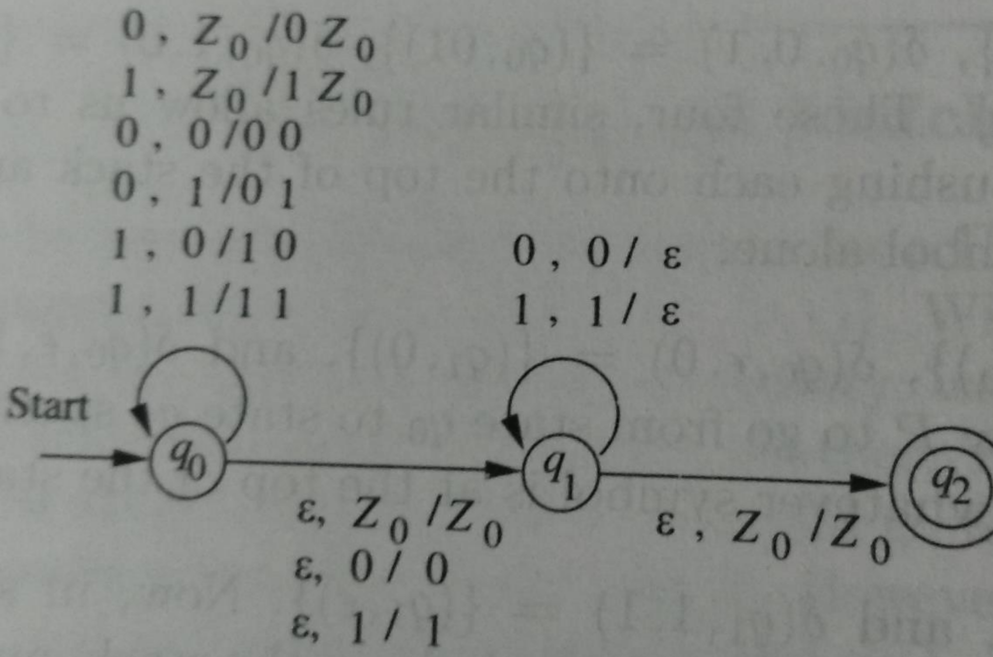
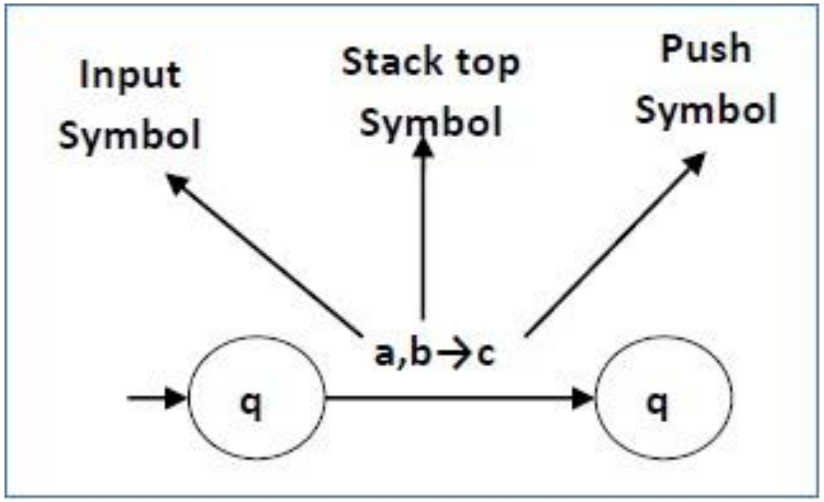


Figure 6.2: Representing a PDA as a generalized transition diagram

Pushdown Automata

- The following diagram shows a transition in a PDA from a state q_1 to state q_2, labelled as a, b → c −

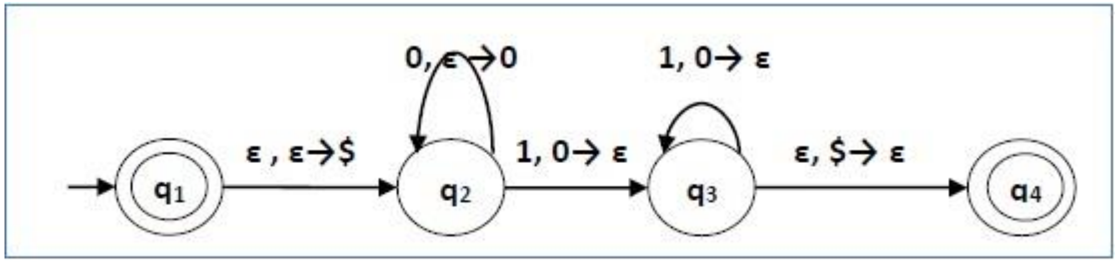


- This means at state $\mathbf{q_1}$, if we encounter an input string **'a'** and top symbol of the stack is **'b'**, then we pop **'b'**, push **'c'** on top of the stack and move to state $\mathbf{q_2}$.

Empty Stack Acceptability

- Here a PDA accepts a string when, after reading the entire string, the PDA has emptied its stack.

- Construct a PDA that accepts **L = {0^n 1^n | n ≥ 0}**



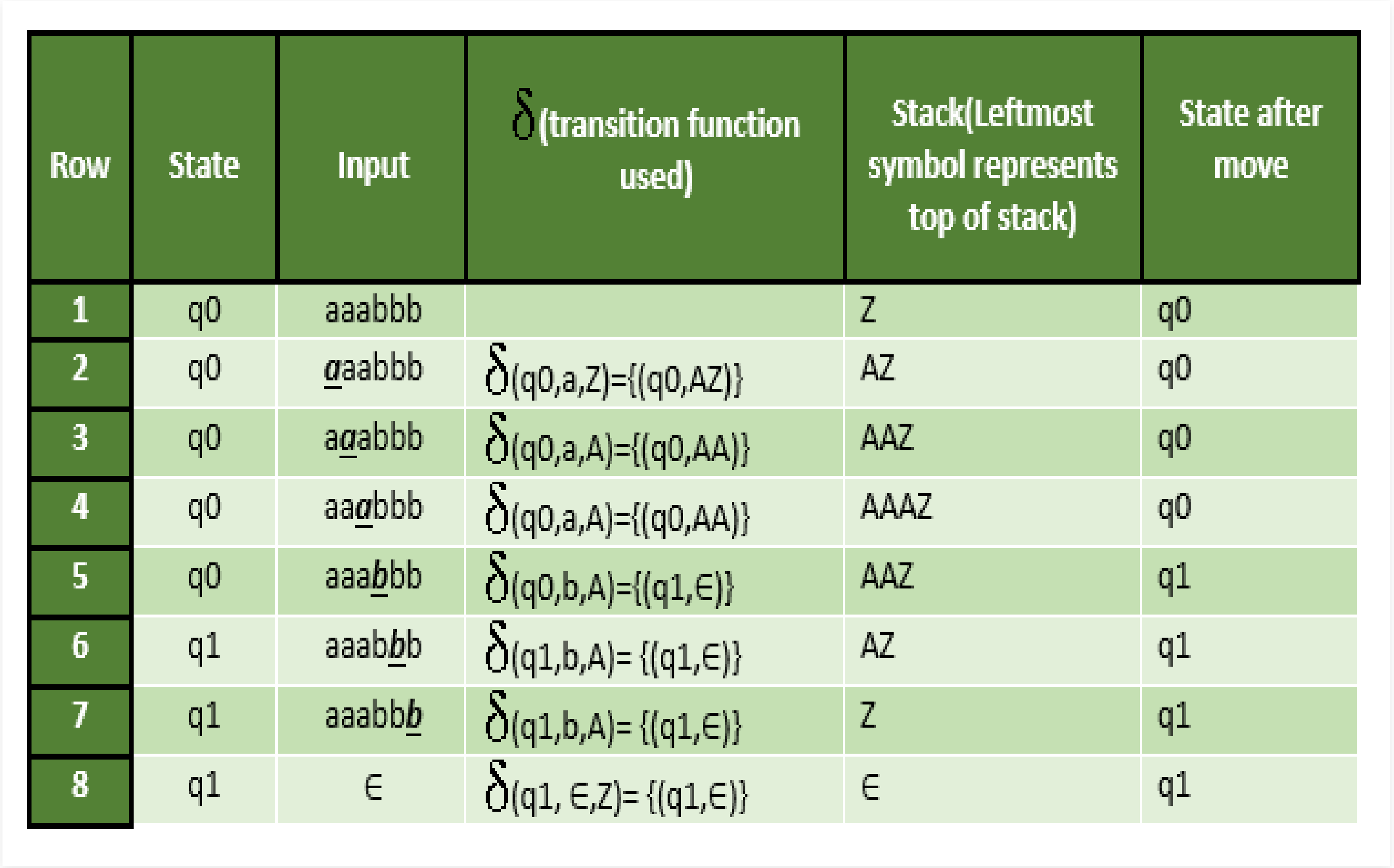
PDA for L= {0^n 1^n | n≥0}

- This language accepts L = {ε, 01, 0011, 000111, ... }

- **Example :** Define the pushdown automata for language {$a^n b^n$ | n > 0}
Solution : M = where Q = { q0, q1 } and Σ = { a, b } and Γ = { A, Z } and &delta is given by :

- &delta(q0, a, Z) = { (q0, AZ) }
&delta(q0, a, A) = { (q0, AA) }
&delta(q0, b, A) = { (q1, ∈) }
&delta(q1, b, A) = { (q1, ∈) }
&delta(q1, ∈, Z) = { (q1, ∈) }

Let us see how this automata works for aaabbb.



Row	State	Input	δ(transition function used)	Stack(Leftmost symbol represents top of stack)	State after move
1	q0	aaabbb		Z	q0
2	q0	*a*aabbb	δ(q0,a,Z)={(q0,AZ)}	AZ	q0
3	q0	a*a*abbb	δ(q0,a,A)={(q0,AA)}	AAZ	q0
4	q0	aa*a*bbb	δ(q0,a,A)={(q0,AA)}	AAAZ	q0
5	q0	aaa*b*bb	δ(q0,b,A)={(q1,∈)}	AAZ	q1
6	q1	aaab*b*b	δ(q1,b,A)= {(q1,∈)}	AZ	q1
7	q1	aaabb*b*	δ(q1,b,A)= {(q1,∈)}	Z	q1
8	q1	∈	δ(q1, ∈,Z)= {(q1,∈)}	∈	q1

Final State Acceptability

- In final state acceptability, a PDA accepts a string when, after reading the entire string, the PDA is in a final state.

- From the starting state, we can make moves that end up in a final state with any stack values. The stack values are irrelevant as long as we end up in a final state.

$$M = (Q=\{q_0, q_1, q_2, q_3\}, \Sigma=\{a,b\}, \Gamma=\{0,1\}, \delta, q_0, z_0 = 0, F=\{q_3\})$$

Where;

$\delta(q_0, a, 0) = \{(q_1, 10), (q_3, \varepsilon)\}$

$\delta(q_0, \varepsilon, 0) = \{(q_3, \varepsilon)\}$

$\delta(q_1, a, 1) = \{(q_1, 11)\}$

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

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

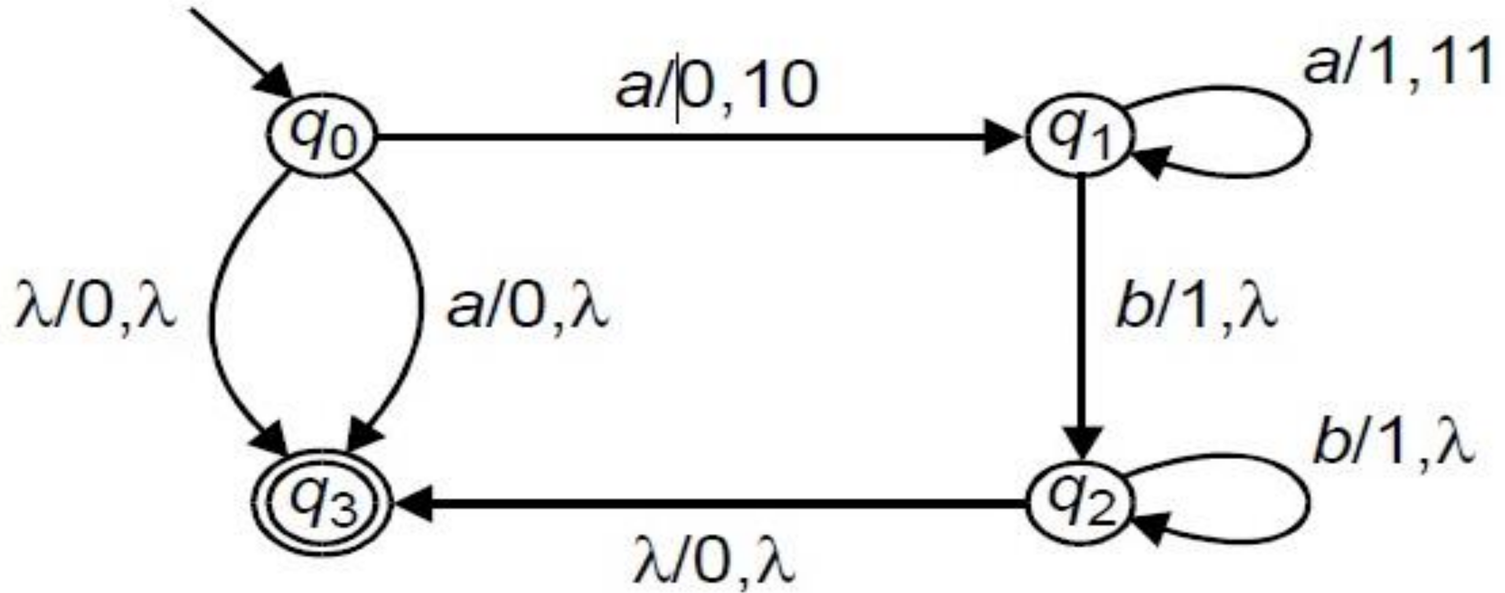
$\delta(q_2, \varepsilon, 0) = \{(q_3, \varepsilon)\}$

$M = (Q=\{q_0, q_1, q_2, q_3\}, \Sigma=\{a,b\}, \Gamma=\{0,1\}, \delta, q_0, z_0 = 0, F=\{q_3\})$

Where;

$\delta(q_0, a, 0) = \{(q_1, 10), (q_3, \varepsilon)\}$
$\delta(q_0, \varepsilon, 0) = \{(q_3, \varepsilon)\}$
$\delta(q_1, a, 1) = \{(q_1, 11)\}$
$\delta(q_1, b, 1) = \{(q_2, \varepsilon)\}$
$\delta(q_1, b, 1) = \{(q_2, \varepsilon)\}$
$\delta(q_2, \varepsilon, 0) = \{(q_3, \varepsilon)\}$

The Pushdown is drawn as follows;



q_0
$a/0,10$
q_1
$a/1,11$
$\lambda/0,\lambda$
$a/0,\lambda$
$b/1,\lambda$
$b/1,\lambda$
q_3
q_2
$\lambda/0,\lambda$