Theoretical Computer Science

Unit 4: Pushdown Automata

Faculty Name : Ms. Namita Pulgam

Index –

Lecture :Deterministic PDA , Non-Deterministic PDA , Application of PDA	3
Lecture : Turing Machine Definition	20
Lecture : Design of TM as generator	

Lecture 40

Deterministic PDA , Non-Deterministic PDA , Application of PDA

Example 3:

Q. Design PDA for L={$a^n b^m c^{m+n}$, $m,n \geq 1$}.

L = { abcc, aabbcccc, aaabbbcccccc,…………………}

Logic : For each input 'a', push ‘X' into stack.

For each input ‘b', push ‘X' into stack.

For each input ‘c', pop one ‘X' from stack

If input is over and stack is empty then accept

$\Sigma = \{ a, b, c \}$

$\Gamma = \{ X, z_0 \}$

States:

q_s : *initial state*

q_0: *read ‘a’ (push)*

q_1:*read ‘b’ (push)*

q_2:*read ‘c’ (pop)*

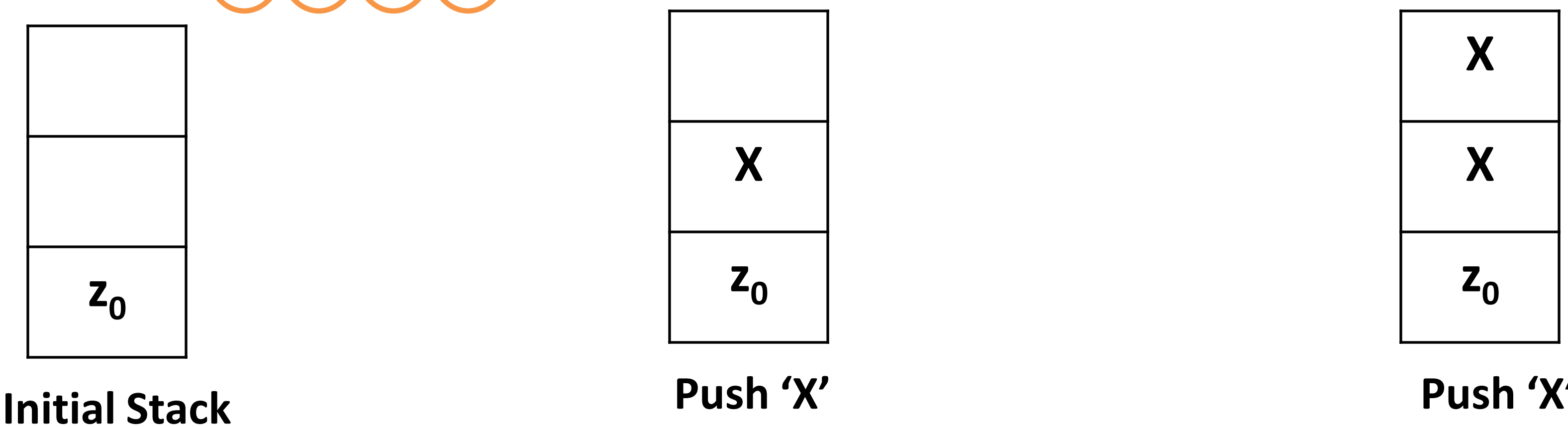
q_3: *input is over and stack is empty (accept)*

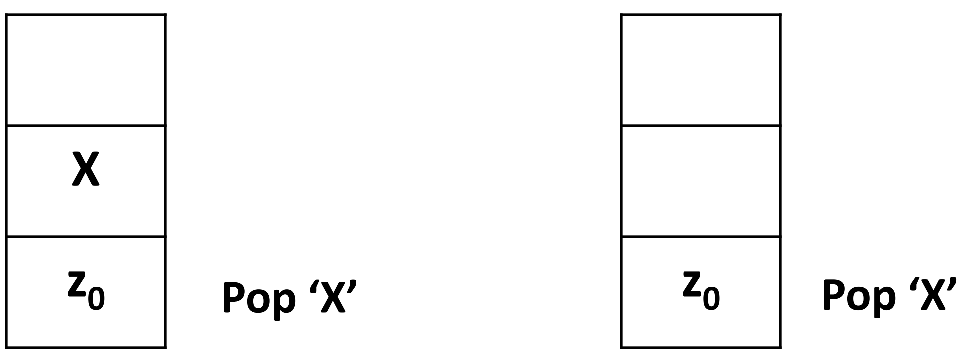
Initial state : q_s

Final state : q_3

Example Processing

Input: a b c c





Stack is empty and input is over so accept string

Transition Rules

$(q_s, a, z_0) = \{ (q_0, Xz_0) \}$ **(First 'a')**

$(q_0, a, X) = \{ (q_0, XX) \}$ **(For remaining 'a')**

$(q_0, b, X) = \{ (q_1, XX) \}$ **(First 'b')**

$(q_1, b, X) = \{ (q_1, XX) \}$ **(For remaining 'b')**

$(q_1, c, X) = \{ (q_2, \varepsilon) \}$ **(First 'c')**

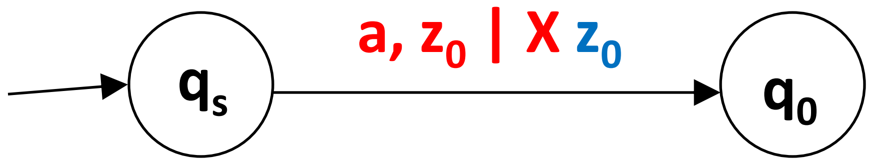
$(q_2, c, X) = \{ (q_2, \varepsilon) \}$ **(For remaining 'c')**

$(q_2, \varepsilon, z_0) = \{ (q_3, z_0) \}$ **(input is over and stack is empty)**

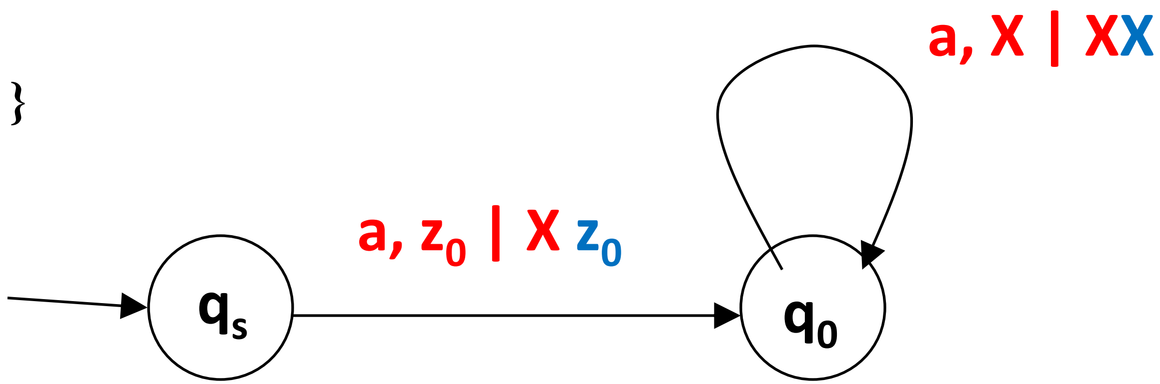
q_s : initial state
q_0: read 'a' (push)
q_1:read 'b' (push)
q_2:read 'c' (pop)
q_3: input is over and stack is empty (accept)

Transition Diagram

$\delta(q_s, a, z_0) = \{ (q_0, Xz_0) \}$

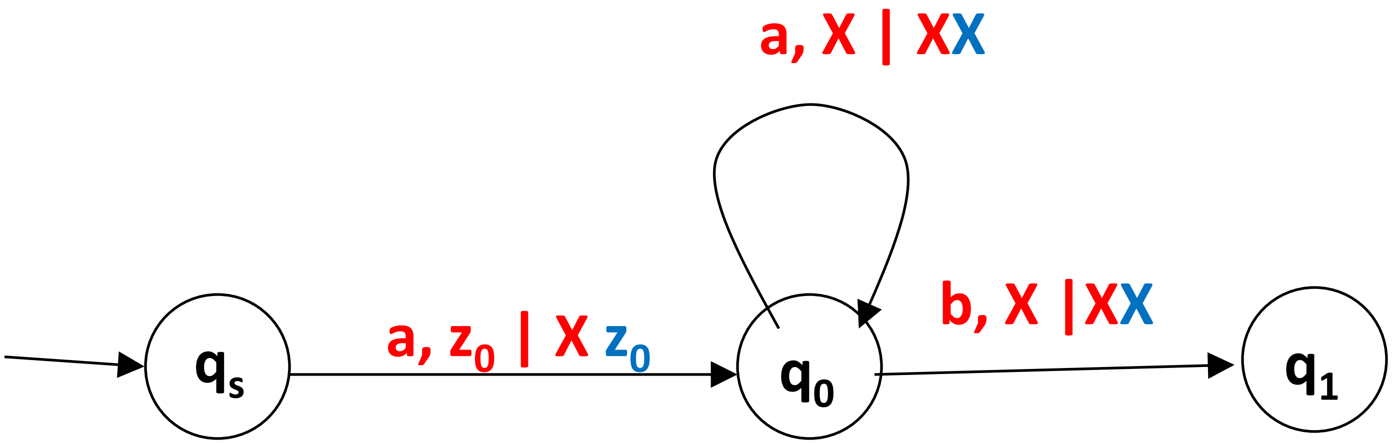


$(q_0, a, X) = \{ (q_0, XX) \}$

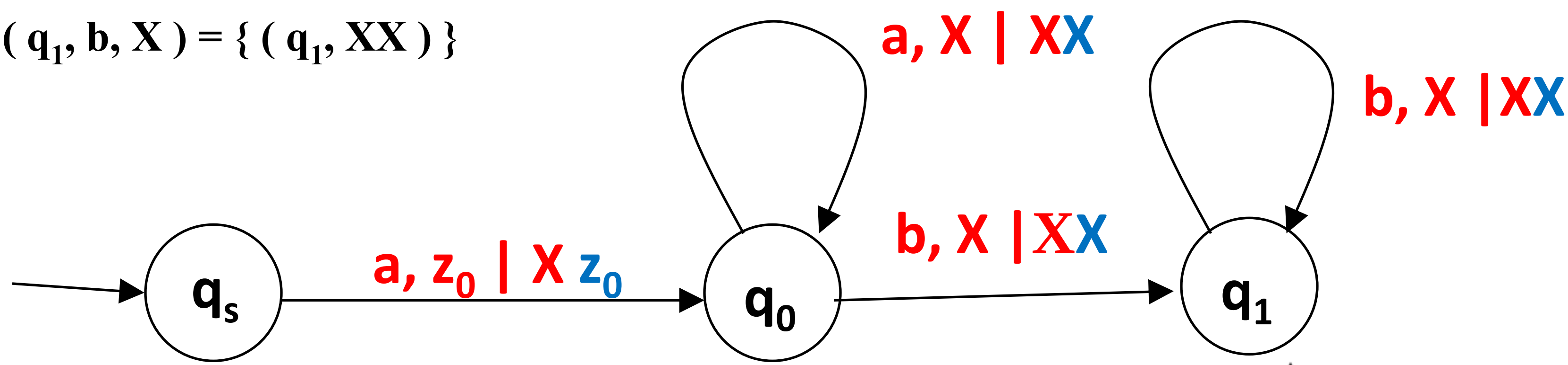


Transition Diagram (cont..)

$(q_0, b, X) = \{ (q_1, XX) \}$



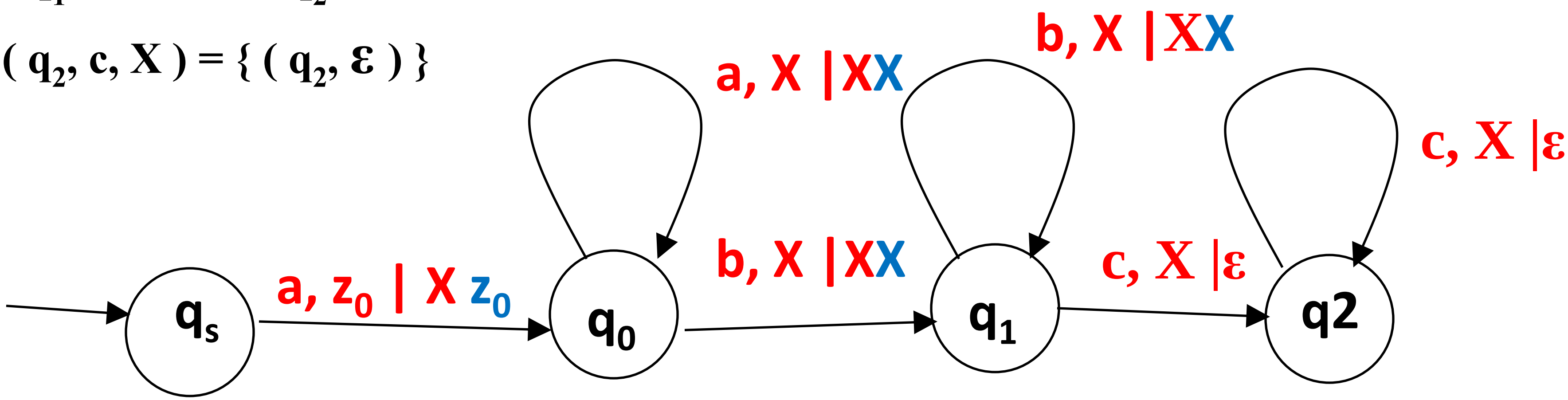
$(q_1, b, X) = \{ (q_1, XX) \}$



Transition Diagram (cont..)

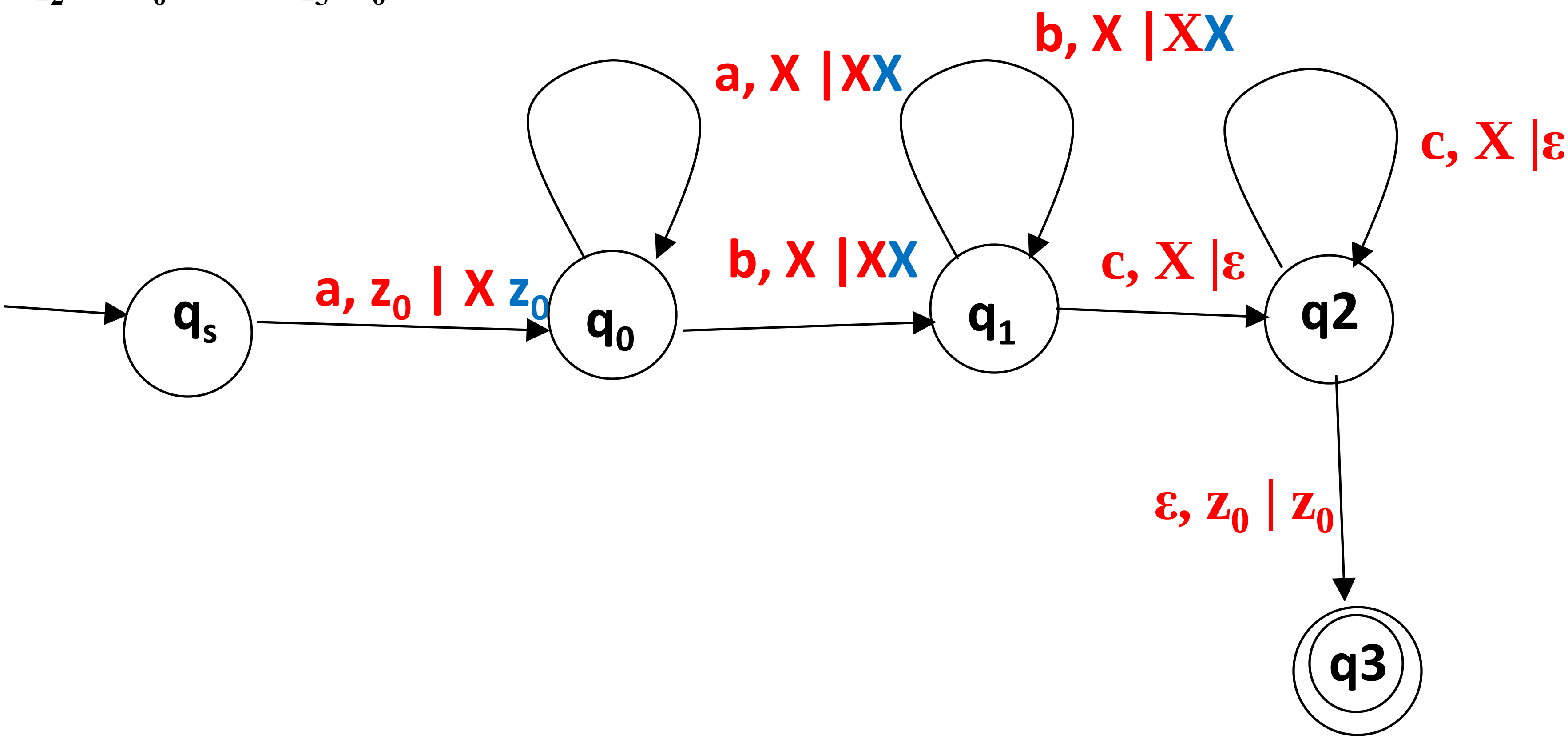
$(q_1, c, X) = \{ (q_2, \varepsilon) \}$

$(q_2, c, X) = \{ (q_2, \varepsilon) \}$



Transition Diagram (cont..)

$(q_2, \varepsilon, z_0) = \{ (q_3, z_0) \}$



Simulation

Input: aabbcccc

(q_s, aabbcccc, z_0)

(q_0, abbcccc, X z_0)

(q_0, bbcccc, XXz_0)

(q_1, bcccc, XXXz_0)

(q_1, cccc, XXXXz_0)

(q_2, ccc, XXXz_0)

(q_2, cc, XXz_0)

(q_2, c, Xz_0)

(q_2, ε, z_0)

(q_3, z_0) accept

Example 4:

Q. Design PDA to recognize L = { $a^n b^m c^n$ $n \geq 1$ }.

Language = { abc, aabcc, aaabbbccc,..... }

Logic : For each input 'a', push ‘a' into stack.

For each input 'b', no operation on stack

For each input ‘c', pop one ‘a' from stack

$\Sigma = \{ a, b, c \}$

$\Gamma = \{ a, Z_0 \}$

States:

q_s : *initial state*

q_0: *read ‘a’ (push)*

q_1:*read ‘b’ (no-operation)*

q_2:*read ‘c’ (pop)*

q_3: *input is over and stack is empty (accept)*

Initial state : q_s

Final state : q_3

Example Processing

Input: a a b b b c c

		a	a	a
	a	a	a	a
z_0	z_0	z_0	z_0	z_0
Initial Stack	**Push 'a'**	**Push 'a'**	**No-operation**	**No-operation**

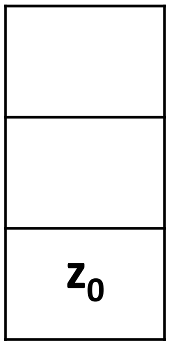
a		
a	a	
z_0	z_0	z_0
No-operation	**Pop 'a'**	**Pop 'a'**

Stack is empty and input is over so accept string

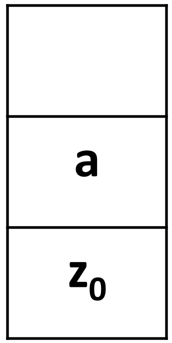
Example Processing

q_s : initial state
q_0: read ‘a’ (push)
q_1:read ‘b’ (no-operation)
q_2:read ‘c’ (pop)
q_3: input is over and stack is empty (accept)

Input: a a b b b c c

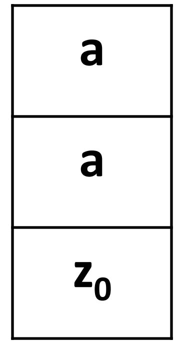


Initial Stack



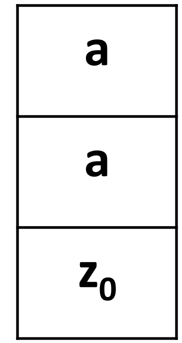
Push ‘a’

$(qs, a, z_0)=\{(q0,az_0)\}$



Push ‘a’

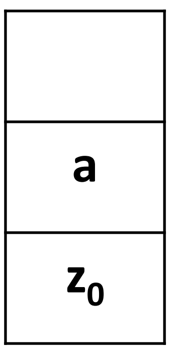
$(q0, a, a)=\{(q0,aa)\}$



No-operation

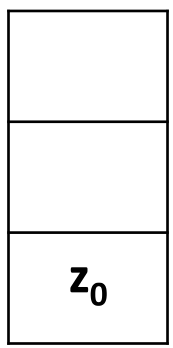
$(q0, b, a)=\{(q1,a)\}$

$(q1, b, a)=\{(q1,a)\}$



Pop ‘a’

$(q1, c, a)=\{(q2,\varepsilon)\}$



Pop ‘a’

$(q2\ c, a)=\{(q2,\varepsilon)\}$

Stack is empty and input is over so accept string

$(q2, \varepsilon, z_0)=\{(q3,z_0)\}$

Final Transition Rules

$(q_s, a, z_0) = \{ (q_0, az_0) \}$

$(q_0, a, a) = \{ (q_0, aa) \}$

$(q_0, b, a) = \{ (q_1, a) \}$

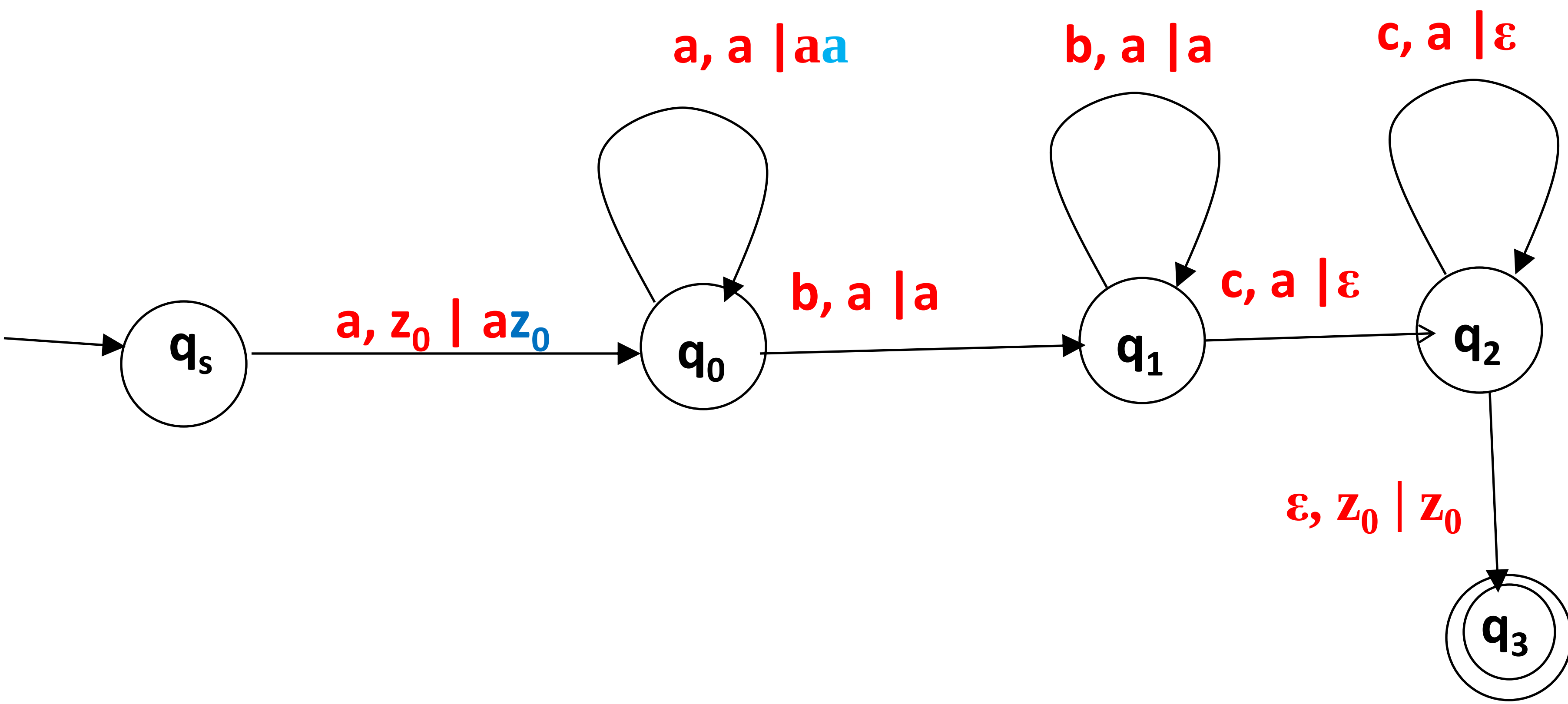
$(q_1, b, a) = \{ (q_1, a) \}$

$(q_1, c, a) = \{ (q_2, \varepsilon) \}$

$(q_2, c, a) = \{ (q_2, \varepsilon) \}$

$(q_2, \varepsilon, z_0) = \{ (q_3, z_0) \}$

Transition Diagram



Simulation

Input: aabbbcc

$(q_s, \textcolor{red}{a}abbbcc, z_0)$

$(q_0, \textcolor{red}{a}bbbcc, az_0)$

$(q_0, \textcolor{red}{b}bbcc, aaz_0)$

$(q_1, \textcolor{red}{b}bcc, aaz_0)$

$(q_1, \textcolor{red}{b}cc, aaz_0)$

$(q_1, \textcolor{red}{c}c, aaz_0)$

$(q_2, \textcolor{red}{c}, az_0)$

(q_2, ε, z_0)

(q_3, z_0) accept

Example 5:

Q. Design PDA for L= {$a^{2n} b^n$, $n \geq 1$}.

L = { aab, aaaabb, aaaaaabbb,…………………}

Logic : Push ‘a' into stack for alternate input ‘a’.

For each input 'b', pop one ‘a' from stack

If input is over and stack is empty then accept

$\Sigma = \{ a, b \}$

$\Gamma = \{ a, z_0 \}$

States :

q_s : *initial state*

q_0: *read ‘a’ (push ‘a’)*

q_1: *read ‘a’ (read ‘a’ , no operation)*

q_2:*read ‘b’ (pop)*

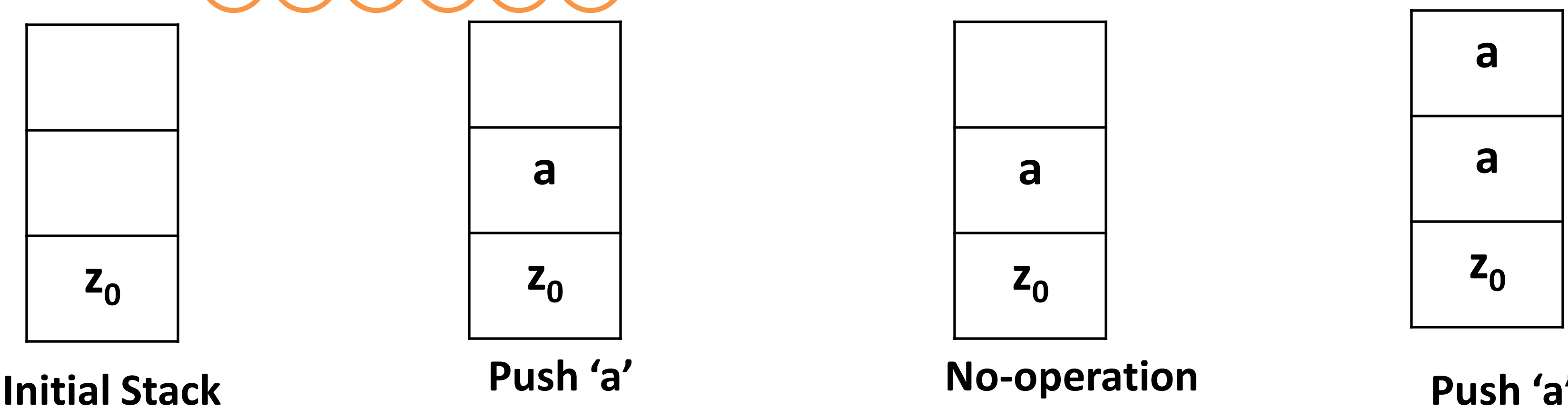
q_3: *input is over and stack is empty (accept)*

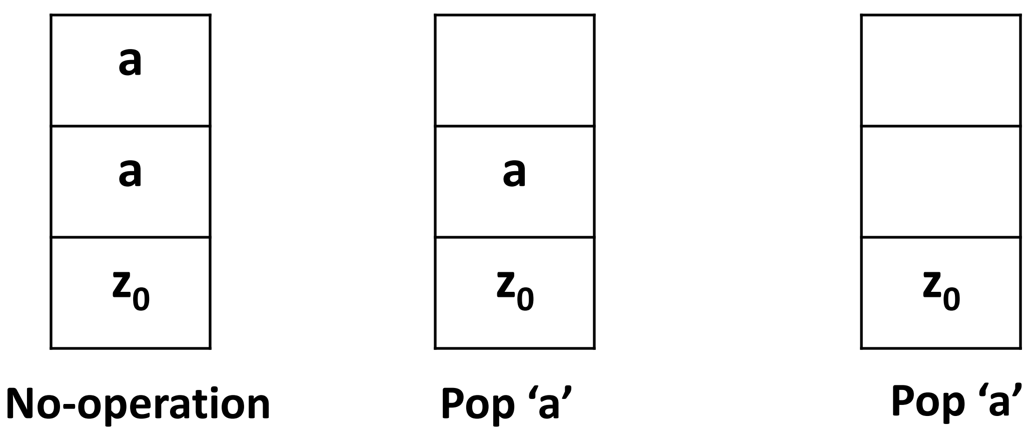
Initial state : q_s

Final state : q_3

Example Processing

Input: a a a a b b

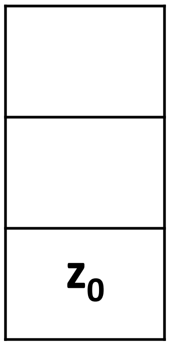




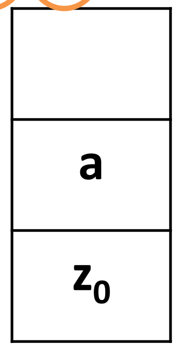
Stack is empty and input is over so accept string

Transition Rules

Input: a a a a b b

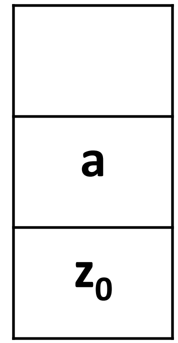


Initial Stack



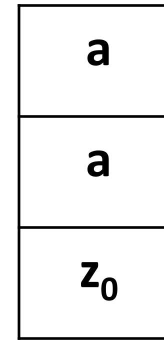
Push ‘a’

$(qs, a, z_0)=\{(q0,az_0)\}$



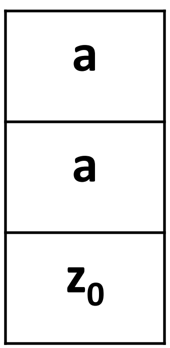
No-operation

$(q0, a, a)=\{(q1,a)\}$



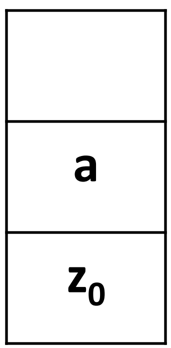
Push ‘a’

$(q1, a, a)=\{(q0,aa)\}$



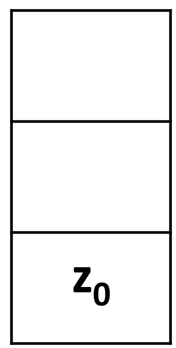
No-operation

$(q0, a, a)=\{(q1,a)\}$



Pop ‘a’

$(q1, b, a)=\{(q2,\varepsilon)\}$



Pop ‘a’

$(q2, b, a)=\{(q2,\varepsilon)\}$

Stack is empty and input is over so accept string

$(q2\ \varepsilon, z_0)=\{(q3,z_0)\}$

Transition Rules

$(qs, a, z_0)=\{(q0,az_0)\}$

$(q0, a, a)=\{(q1,a)\}$

$(q1, a, a)=\{(q0,aa)\}$

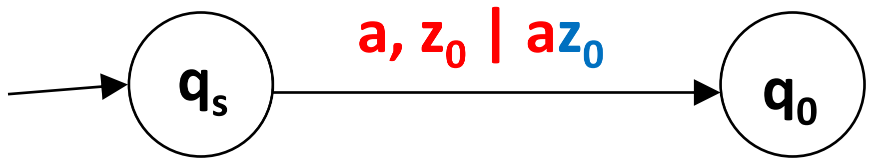
$(q1, b, a)=\{(q2,\varepsilon)\}$

$(q2, b, a)=\{(q2,\varepsilon)\}$

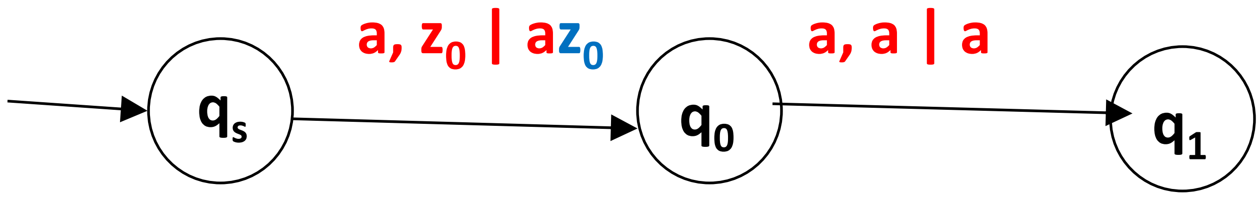
$(q2, \varepsilon, z_0)=\{(q3,z_0)\}$

Transition Diagram

(qs, a, z_0)={(q0,az_0)}

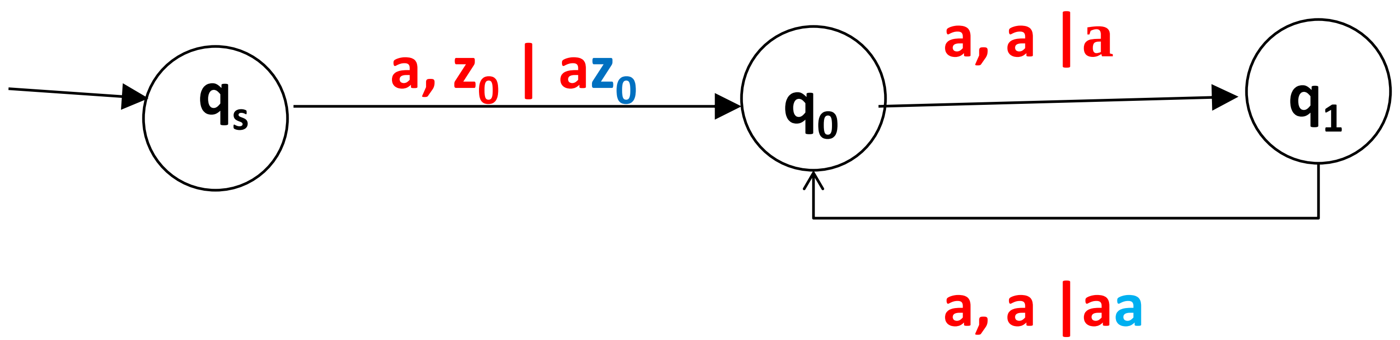


(q0, a, a)={(q0,a)}



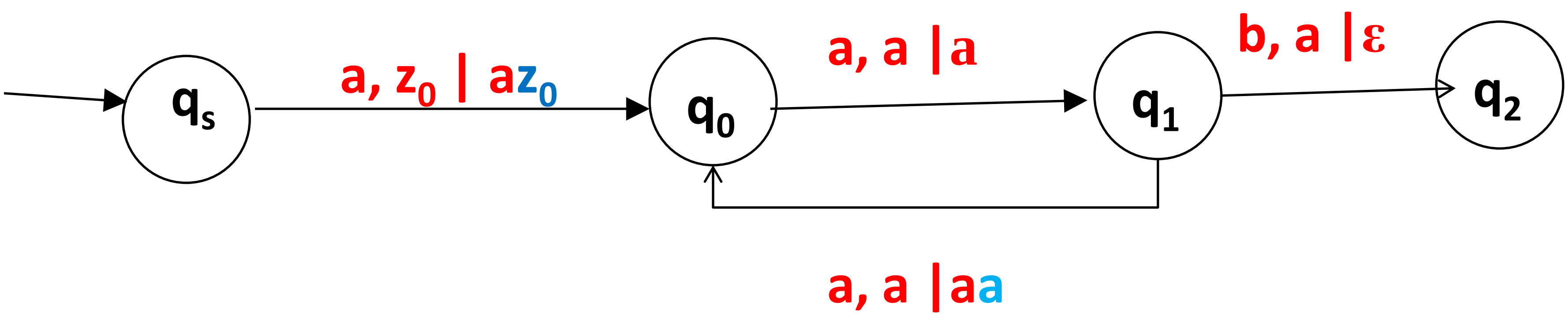
Transition Diagram (cont..)

(q1, a, a)={(q0,aa)}



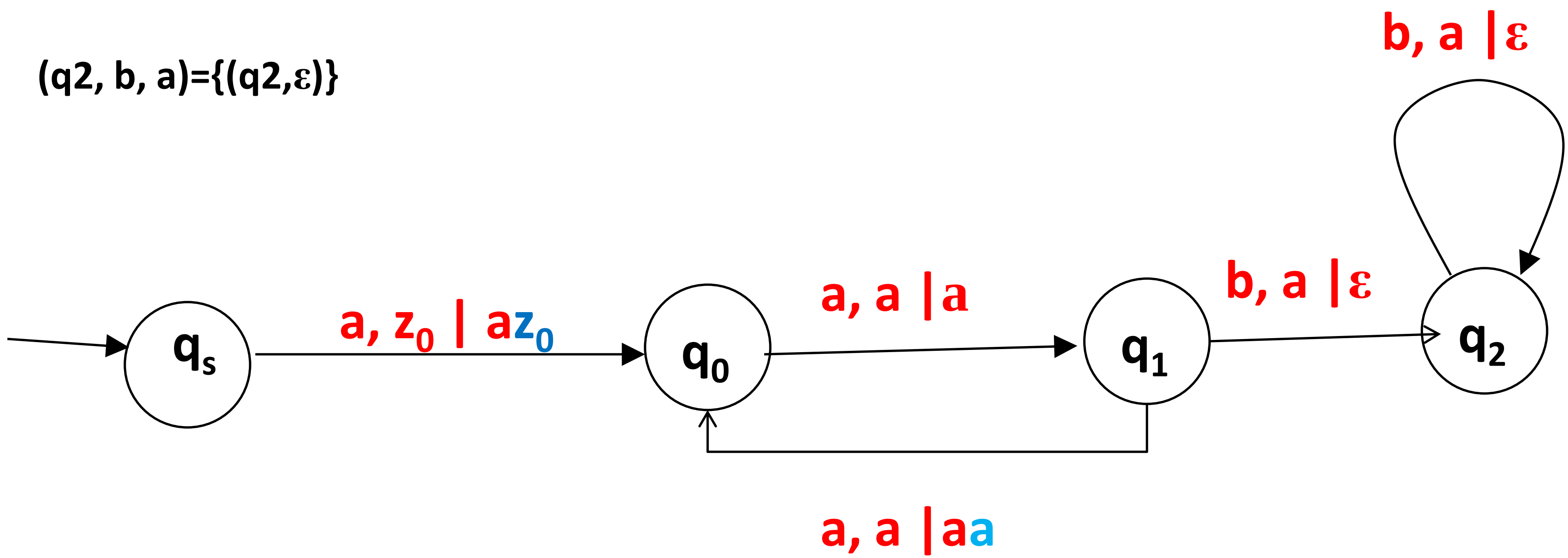
Transition Diagram (cont..)

(q1, b, a)={(q2,ε)}



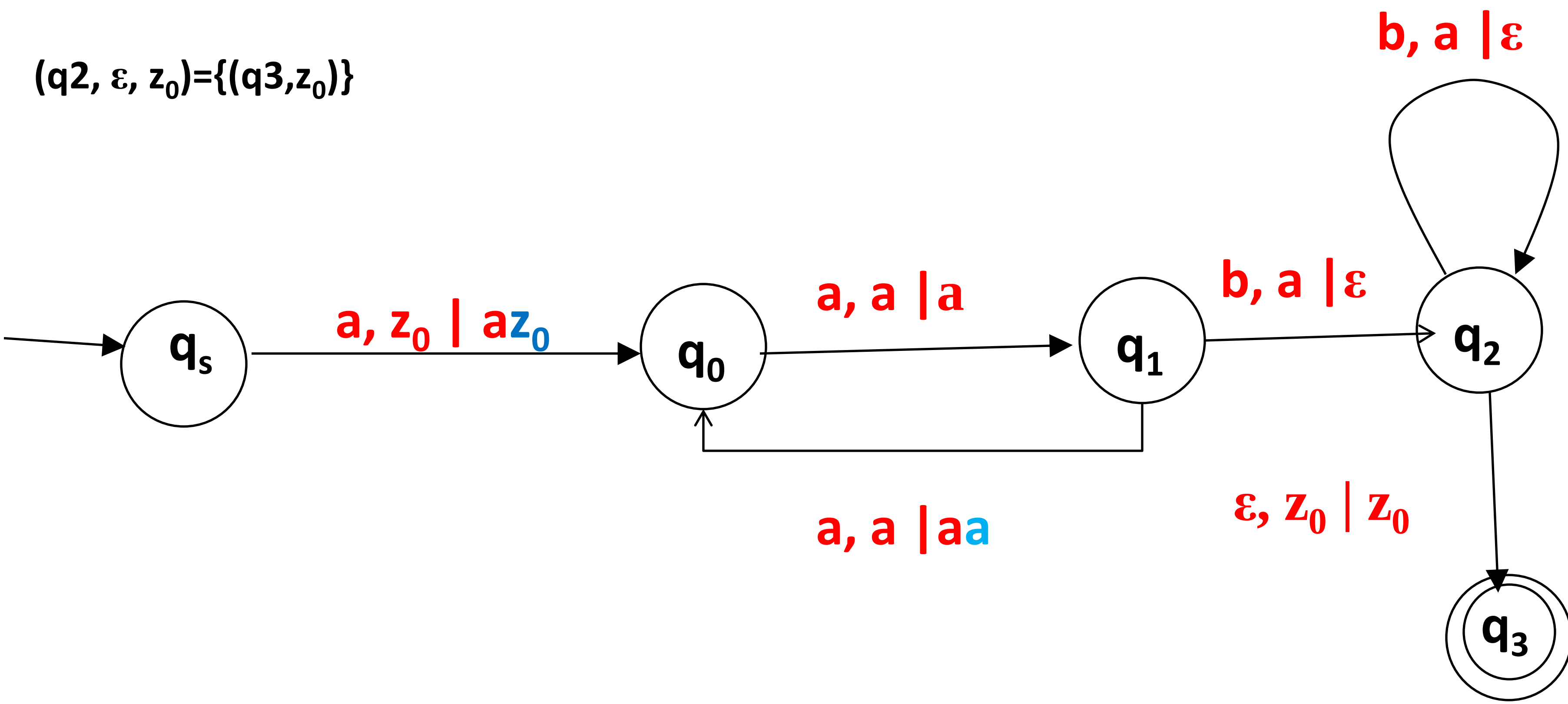
Transition Diagram (cont..)

(q2, b, a)={(q2,ε)}



Transition Diagram (cont..)

$(q2, \varepsilon, z_0)=\{(q3,z_0)\}$



Simulation

Input: aaaabb

(qs, aaaabb, z_0)

(q0, aaabb, a z_0)

(q1, aabb, a z_0)

(q0, abb, aa z_0)

(q1, bb, aa z_0)

(q2, b, az_0)

(q2, ε, z_0)

(q3, z_0) accept

Example 6:

Q. Design PDA to recognize L={a^n b^m |$n < m$}.

Language = {abbb, aabbb, aaabbbbb,.....}

Logic : For each input 'a', push ‘a' into stack.

For each input 'b', pop one ‘a' from stack

Stack will be empty before input is over .

$\sum = \{a, b\}$

$\Gamma = \{a, Z_0\}$

States:

q_s : *initial state*

q_0: *read ‘a’ (push)*

q_1:*read ‘b’ (pop)*

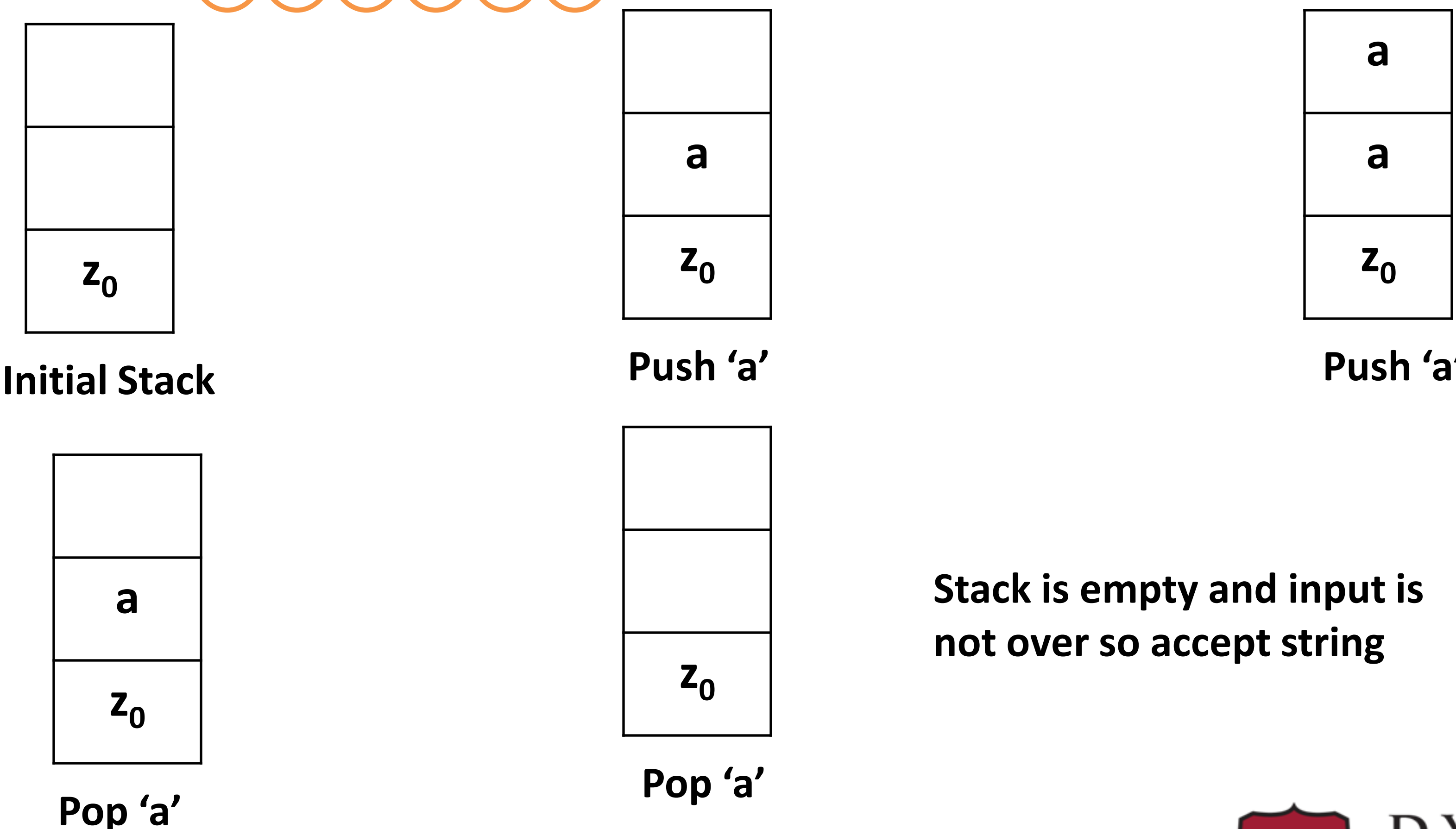
q_2: *input is not over and stack is empty (accept)*

Initial state : q_s

Final state: q_2

Example Processing

Input: a a b b b b

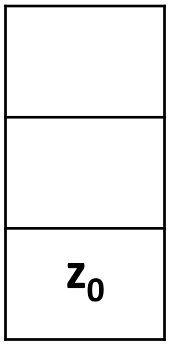


Stack is empty and input is not over so accept string

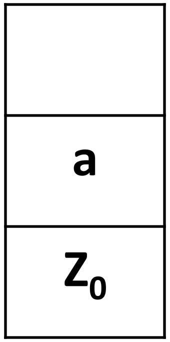
Transition Rules

q_s : initial state
q_0: read ‘a’ (push)
q_1:read ‘b’ (pop)
q_2: input is not over and stack is empty (accept)

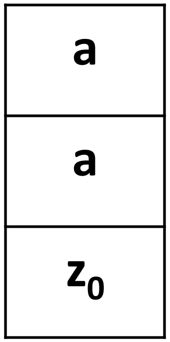
Input: a a b b b b



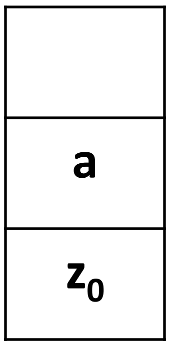
Initial Stack : Push ‘a’
(qs, a, z0)={(q0,az_0)}



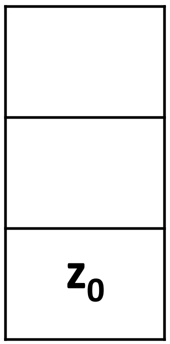
Push ‘a’
(q0, a, a)={(q0,aa)}



Pop ‘a’
(q0, b, a)={(q1,ε)}



Pop ‘a’
(q1, b, a)={(q1,ε)}



(q1, b, z_0)={(q2,z_0)}

Stack is empty and input is not over so accept string

Transition Rules

$(qs, a, z_0)=\{(q0,az_0)\}$

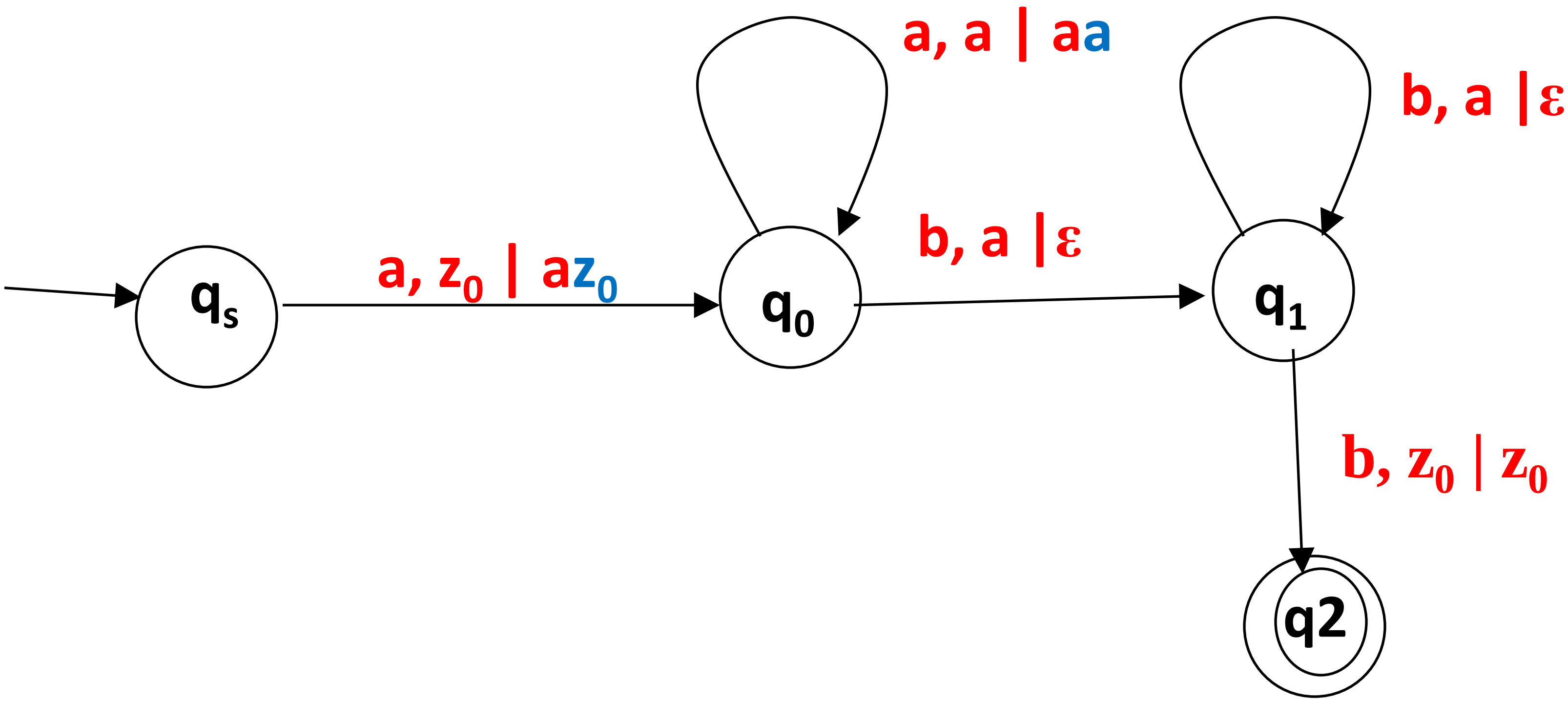
$(q0, a, a)=\{(q0,aa)\}$

$(q0, b, a)=\{(q1,\varepsilon)\}$

$(q1, b, a)=\{(q1,\varepsilon)\}$

$(q1, b, z_0)=\{(q2,z_0)\}$

Transition Diagram



Simulation

Input: aabbb

(qs, aabbb, z_0)

(q0, abbb, az_0)

(q0, bbb, aaz_0)

(q1, b, az_0)

(q1, b, z_0)

(q2, z_0) accept

Example 7 :

Q. Design PDA to recognize L = {WcW^T |$W \in (a+b)^*$}

Let us consider W is a string of length 'n' then W^T is a reverse string of W. The string WcW^T is string of odd length with middle character c. It is a palindrome.

Language = { abcba, aacaa, bacab,.....}

Logic : Push first 'n' symbols on stack

For 'c' no operation on stack

For next 'n' symbols pop one symbol from stack if match found

$\Sigma = \{a, b, c\}$

$\Gamma = \{a, b, Z_0\}$

States: q_s : *initial state*

q_0: *read 'a' or 'b' (push)*

q_1:*read 'c' (no-operation)*

q_2:*read 'a' and stack top is 'a' (pop)*

read 'b' and stack top is 'b' (pop)

q_3: *input is over and stack is empty (accept)*

Initial state : q_s

Final state : q_3

Example Processing

Input: a b c b a

z_0

Initial Stack

a
z_0

Push 'a'

b
a
z_0

Push 'b'

b
a
z_0

No-operation

a
z_0

Input is 'b' and stack top is 'b' so Pop 'b'

z_0

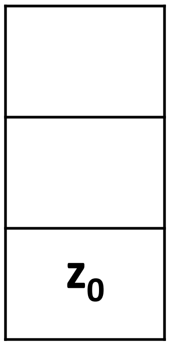
Input is 'a' and stack top is 'a' so Pop 'a'

Stack is empty and input is over so accept string

Transition Rules

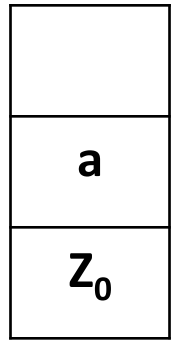
q_s : initial state, q_0: read ‘a’ or ‘b’ (push),
q_1:read ‘c’ (no-operation)
q_2:read ‘a’ and stack top is ‘a’ (pop)
read ‘b’ and stack top is ‘b’ (pop)
q_3: input is over and stack is empty (accept)

Input: a b c b a



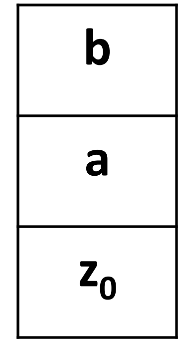
Initial Stack : Push ‘a’

(qs, a, z0)={(q0,az_0)}



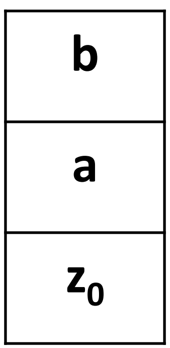
Push ‘b’

(q0, b, a)={(q0,ba)}



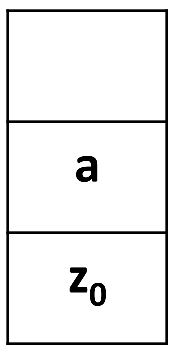
No-operation

(q0, c, b)={(q1,b)}



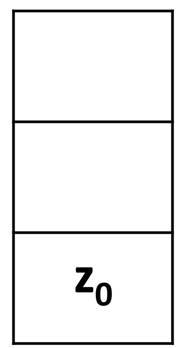
Pop ‘b’

(q1, b, b)={(q2,ε)}



Pop ‘a’

(q2, a, a)={(q2, ε)}



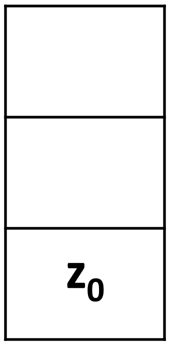
(q2, ε , z_0)={(q3, z_0)}

Stack is empty and input is over so accept string

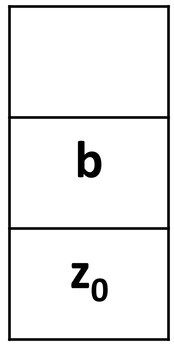
D Y PATIL DEEMED TO BE UNIVERSITY RAMRAO ADIK INSTITUTE OF TECHNOLOGY NAVI MUMBAI

Example Processing

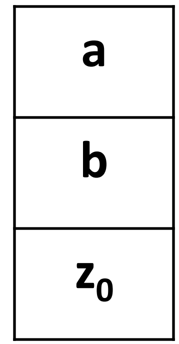
Input: b a c a b



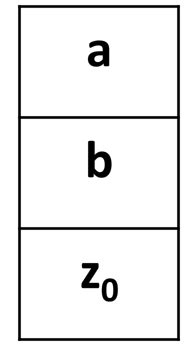
Initial Stack



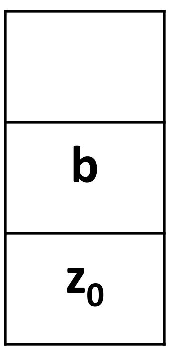
Push ‘a’



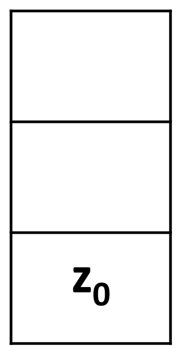
Push ‘b’



No-operation



Input is ‘a’ and stack top is ‘a’ so Pop ‘a’



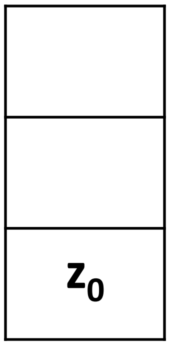
Input is ‘b’ and stack top is ‘b’ so Pop ‘b’

Stack is empty and input is over so accept string

Transition Rules

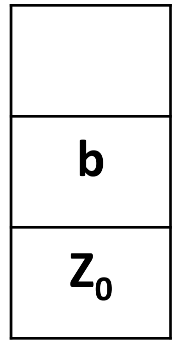
q_s : initial state, q_0: read 'a' or 'b' (push),
q_1:read 'c' (no-operation)
q_2:read 'a' and stack top is 'a' (pop)
read 'b' and stack top is 'b' (pop)
q_3: input is over and stack is empty (accept)

Input: b a c a b



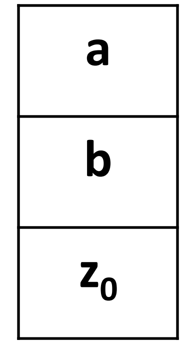
Initial Stack : Push 'b'

(qs, b, z0)={(q0,bz_0)}



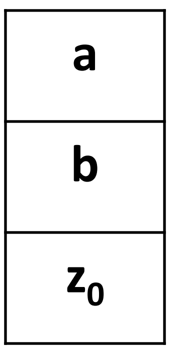
Push 'a'

(q0, a, b)={(q0,ab)}



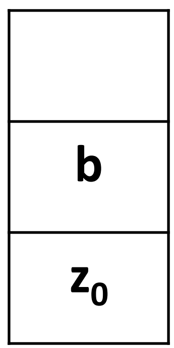
No-operation

(q0, c, a)={(q1,a)}



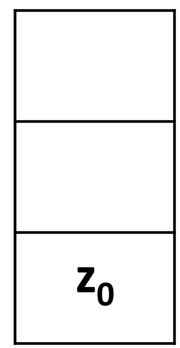
Pop 'a'

(q1, a, a)={(q2,ε)}



Pop 'b'

(q2, b, b)={(q2, ε)}



(q2, ε , z_0)={(q3, z_0)}

Stack is empty and input is over so accept string

Transition Rules

$(qs, a, z_0) = \{ (q0, az_0) \}$

$(qs, b, z_0) = \{ (q0, bz_0) \}$

$(q0 , a, b) = \{ (q0, ab) \}$

$(q0, b, a) = \{ (q0,ba) \}$

$(q0, a, a) = \{ (q0, aa) \}$

$(q0, b, b) = \{ (q0, bb) \}$

$(q0, c, a) = \{ (q1, a) \}$

$(q0, c, b) = \{ (q1, b) \}$

$(q1, a, a) = \{ (q2, \varepsilon) \}$

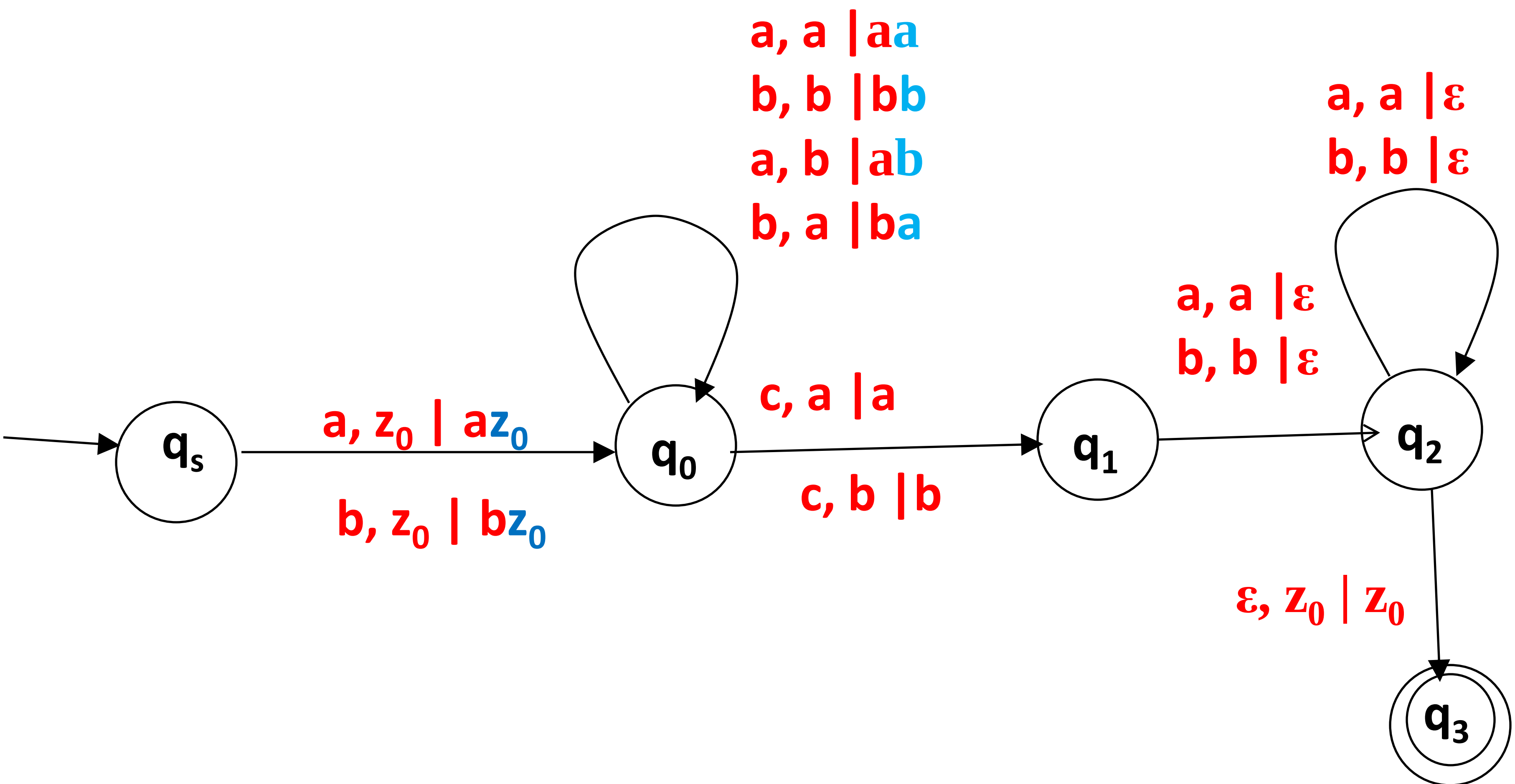
$(q1, b, b) = \{ (q2, \varepsilon) \}$

$(q2, a, a) = \{ (q2, \varepsilon) \}$

$(q2, b, b) = \{ (q2, \varepsilon) \}$

$(q2, \varepsilon, z_0) = \{ (q3, z_0) \}$

Transition Diagram



Simulation

Input: aabcbaa

(qs, aabcbaa, z_0)

(q0, abcbaa, az_0)

(q0, bcbaa, aaz_0)

(q0, cbaa, $baaz_0$)

(q1, baa, $baaz_0$)

(q2, aa, aaz_0)

(q2, a, az_0)

(q2, ε, z_0)

(q3, z_0) accept

Types of Pushdown Automata

- Similar to Finite automata, there are two types of push down automata namely: Deterministic PDA and Non-deterministic PDA
- In deterministic PDA, there is only one move in every situation while in Non-deterministic PDA, there can be multiple moves under a situation.
- The language accepted by deterministic PDA lies between regular language and context free language as in every situation there is only one move.
- A DPDA is defined as:

  $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

  where

  δ (q, a, X) has one move for every $q \in Q$, $X \in \Gamma$ and $a \in \Sigma$

Continued..

- Every context free language cannot be recognized by DPDA but for every regular language we can design a DPDA.

  Example: {abba, aa, ..} string of the form WW^R cannot be recognized by DPDA.

- The strings of the form WW^R *generate even length palindromes. e.g. abba, baab.*
- For such strings there is no way to decide the mid position in string. So we cannot decide how many symbols to be pushed onto the stack and when to pop from stack.
- Thus a string of the form WW^R *cannot be recognized by DPDA.*
- *The NPDA can recognize such language.*
- **A NPDA is defined as:**

  $M= (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ where

  $\delta(q, a, X)$ has one move for every $Q \in 2^Q$, $X \in \Gamma$ and $a \in \Sigma^*$

DPDA vs NPDA

Deterministic PDA	Non deterministic PDA
There is only one move in every situation	There can be multiple moves under a situation
DPDA is less powerful than NPDA	NPDA is powerful than DPDA
Every context free language cannot be recognized by DPDA. The language accepted by DPDA lies between a regular language and CFL.	Every context free language can be recognized by NPDA
A even length palindrome cannot be accepted by DPDA	A even length palindrome can be accepted by NPDA

Applications

- PDA can be used to determine whether a particular string is derived from the start symbol of CFG.
- This kind of functionality is required in parsing phase of compiler. Parsing is also called as syntax analysis.
- It is used to recognize the sentence in a particular language. It discovers structure of a program and constructs a tree to represent this structure. This tree is called as parse tree.
- The parse tree generated by parser is further used by the code generator to produce the target code.

Thank You

Lecture No 43:

Definition, Transitions (Diagrams, Functions and Tables) Turing Machine

Basics

- Finite automata is mathematical model of finite state machine which just plays the role of language acceptor.
- FSM can not remember an arbitrarily long sequence of symbols.
- It has a read head that can move only in one direction to the right always.
- It can not move backwards to retrieve previous information stored onto the tape.
- FSM can not check if a set of parentheses are well formed or for palindrome sequences that need to store data to be used for later computation.
- All these limitations arise because FSMs do not have memory and hence they can not solve problems that need to store data to be used for later computation.
- Some of these limitations are overcome by the pushdown automata(PDA).

Basics (cont..)

- As PDA has memory in terms of stack, it can remember the sequence of symbols.
- PDA is more powerful than FA, still has some limitations like it cannot recognize string of even length palindrome.
- PDA also scan input only in one direction.
- Thus, these limitations impose the need for more powerful machine.
- This requirement is satisfied by new model designed by Alan Turing in 1960s.
- Alan Turing is father of such model which has computing capability of general purpose computer.
- This model is popularly known as Turing Machine.

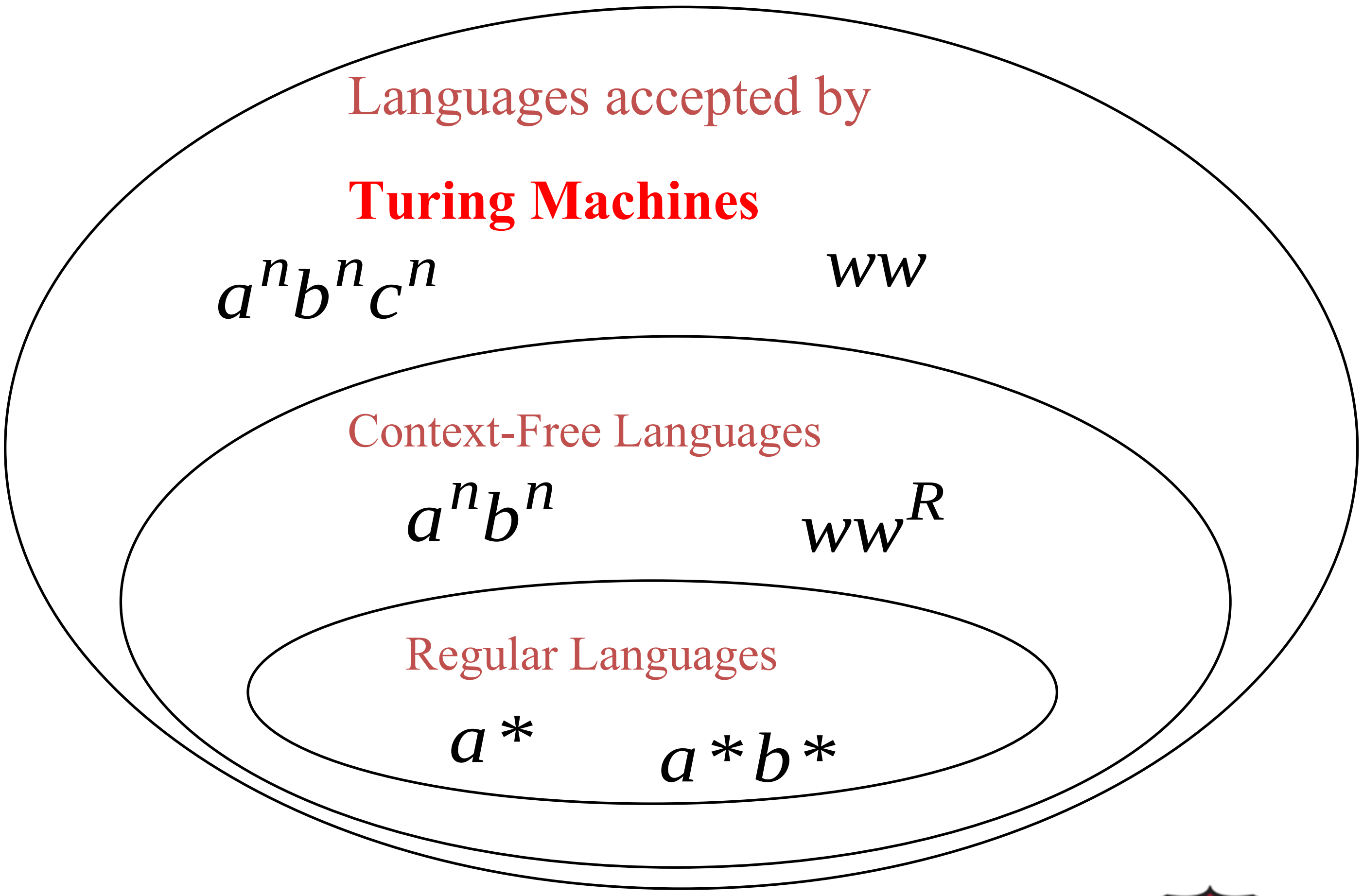
Introduction to Turing Machine

1. It has external memory(input tape) which remembers arbitrarily long sequence of input.
2. It has unlimited memory capability.
3. The head is Read or Write head so it has capability to write the output on its tape.
4. The read / write head can move in left and right both directions.
5. The machine can produce certain output based on its input. Sometimes it may be required that the same input has to be used to generate the output. So in this machine the distinction between input and output has been removed. Thus a common set of alphabets can be used to the Turing machine.
6. It is not just the language acceptor/recognizer but also performs basic computations like addition, subtraction, multiplication and so on.

Differences between finite automata and Turing machine

1. A Turing machine can both write on the tape and read from it.
2. The read-write head can move both to the left and to the right.
3. The tape is infinite.
4. The special states for rejecting and accepting take immediate effect.

The Language Hierarchy



Formal Definition

A Turing Machine is represented by a 7-tuple s

$$\mathbf{T = \{ Q, \Sigma, \Gamma, \delta, q_0, B, F \}}$$

Where,

Q : Finite set of states

Σ : Finite set of input symbols

Γ : Finite set of tape symbols which include the blank symbol

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function

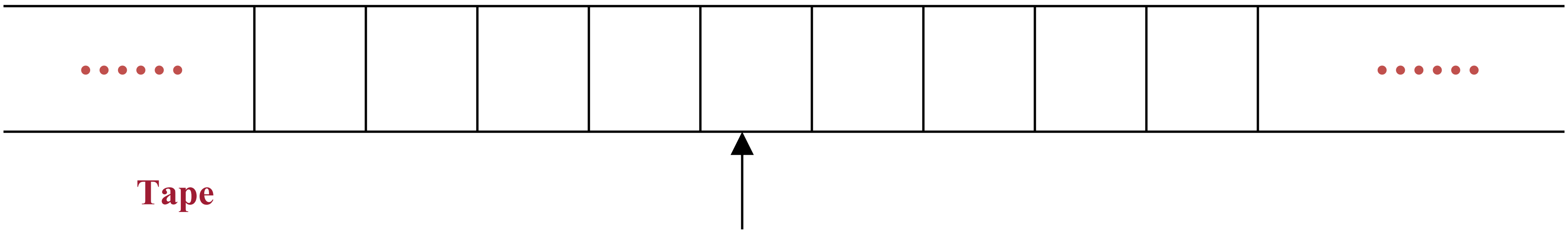
q_0 : Initial State $\in Q$

B : Blank Symbol

F : Finite set of final states

A Turing Machine

No boundaries -- infinite length

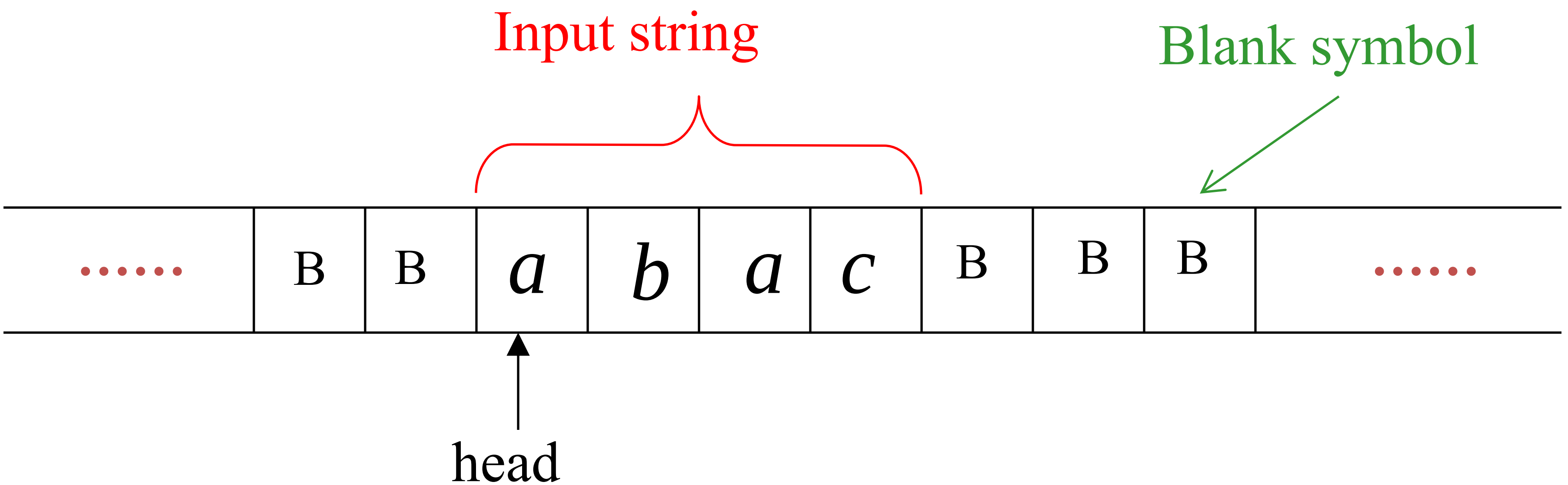


Tape

Read-Write head

The head moves Left or Right

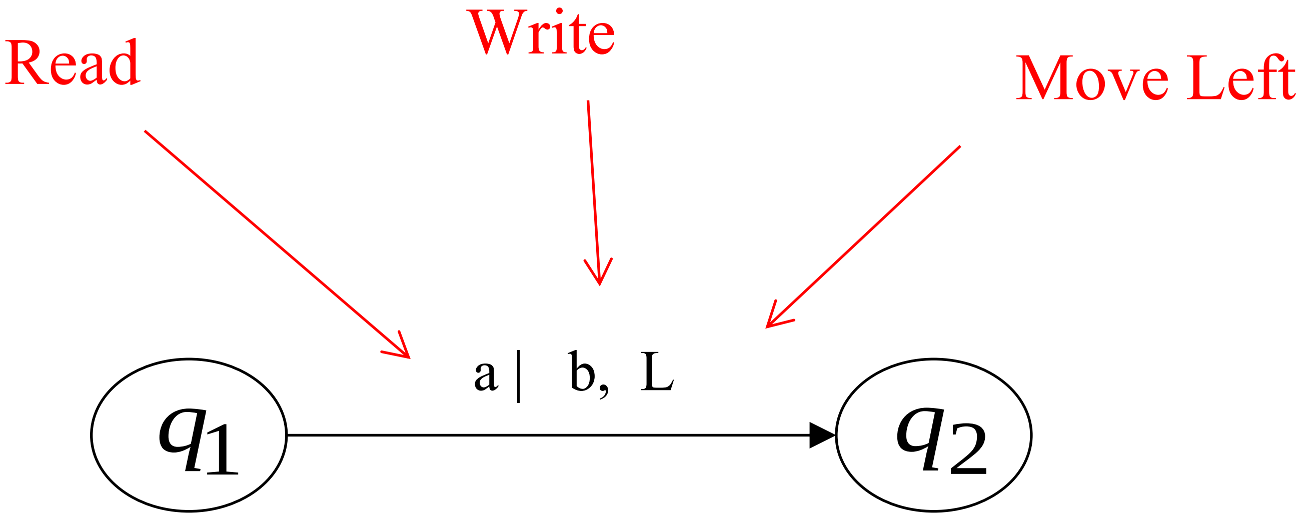
The Input String

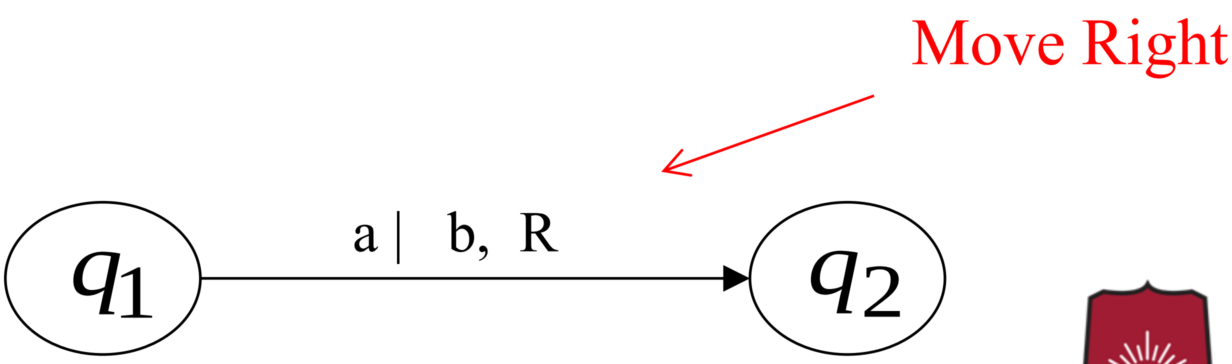


Head starts at the leftmost position

of the input string

Representation of Transitions





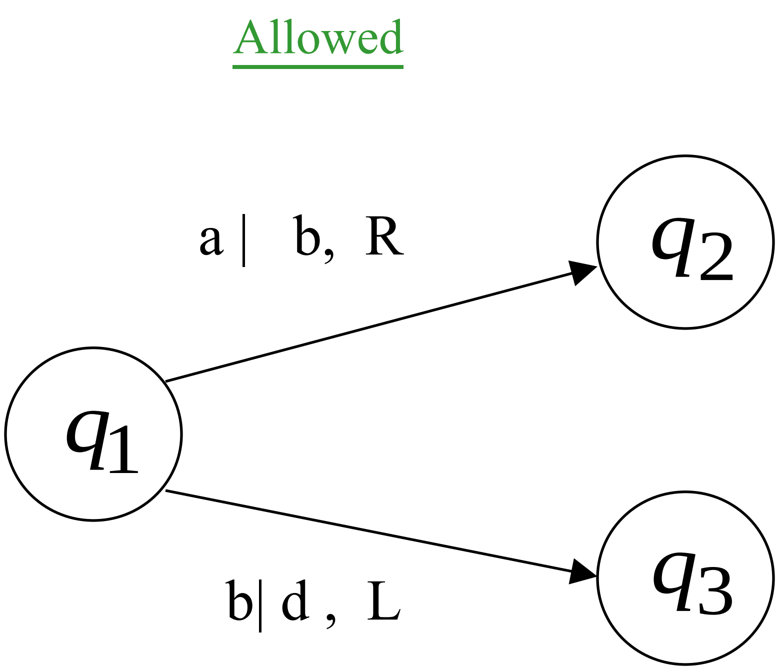
Transitions

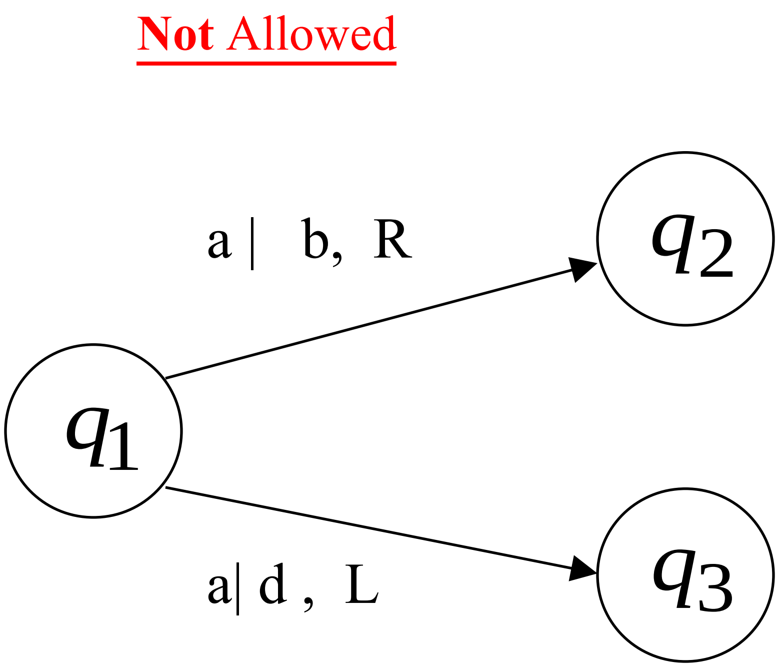
- The transition of Turing machine depends on the input symbol and current state.
- It reads one symbol from the tape and moves either to left or right or remains in the same position.
- The symbol under head is either replaced by new symbol or kept as it is.
- Consider following transition:

$$\delta\,(\underbrace{q_0}_{\text{Current State}},\ \underbrace{a}_{\text{Current Input}}) \;=\; \{\underbrace{q_1}_{\text{Next State}},\ \underbrace{b}_{\text{Replaced Symbol}},\ \underbrace{R}_{\text{Move}}\}$$

Determinism

Turing Machines are deterministic





Acceptance

Accept Input String

If machine halts

in an accept state

Reject Input String

If machine halts

in a non-accept state

or

If machine enters

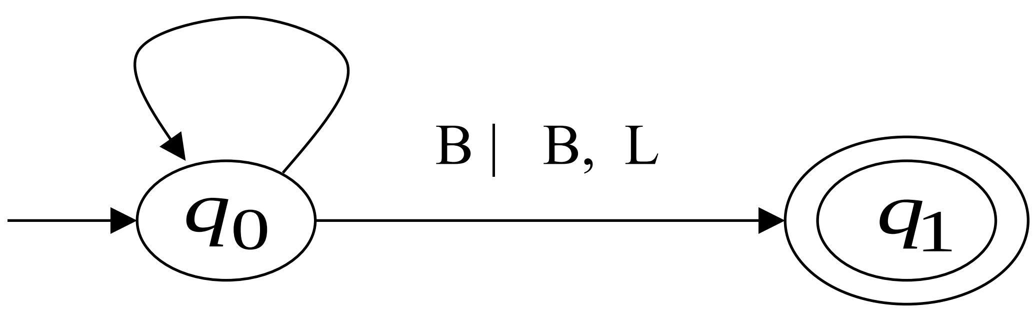
an *infinite loop*

In order to accept an input string, it is not necessary to scan all the symbols in the string

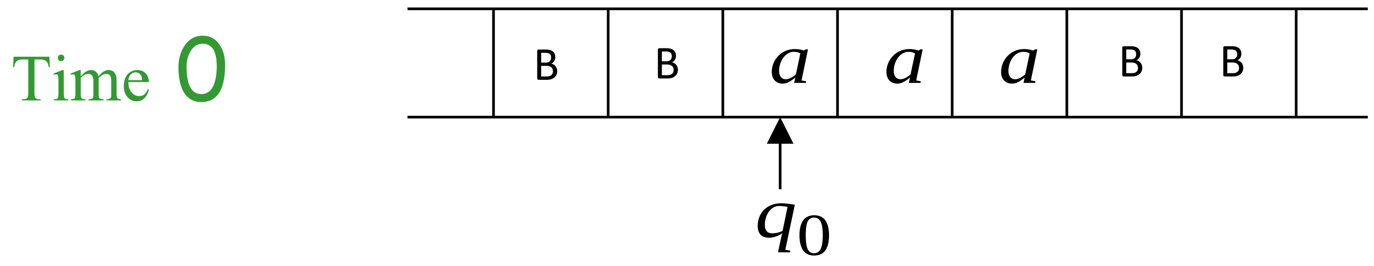
Turing Machine Example

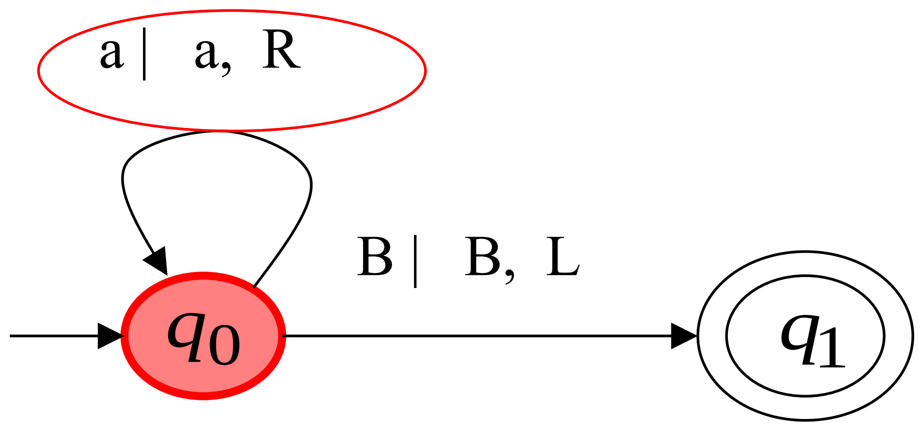
Input alphabet $\Sigma = \{a, b\}$

Accepts the language: a^*



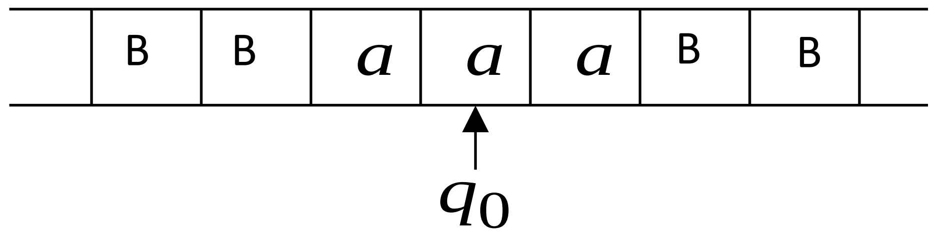
Turing Machine Example (cont..)

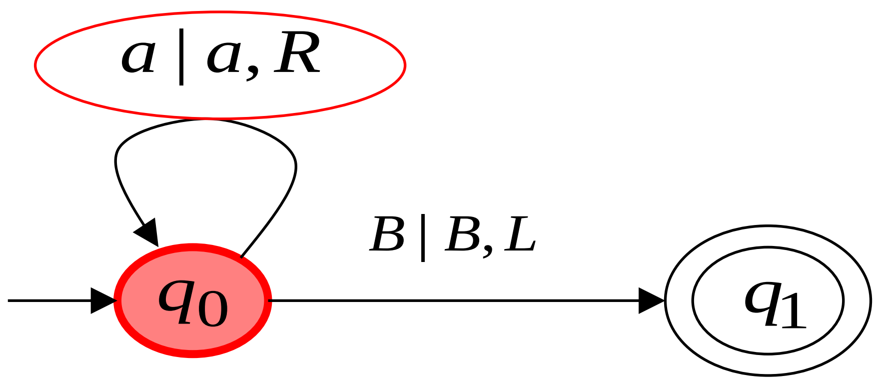




Turing Machine Example (cont..)

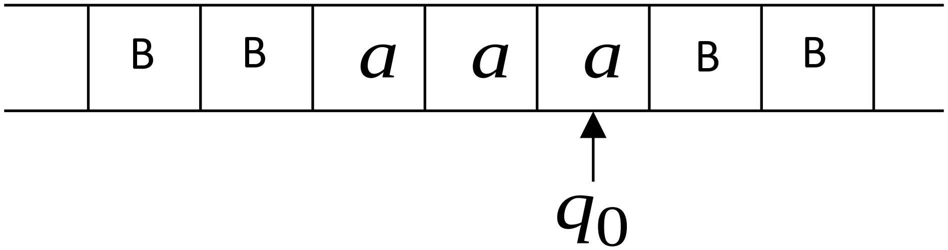
Time 1

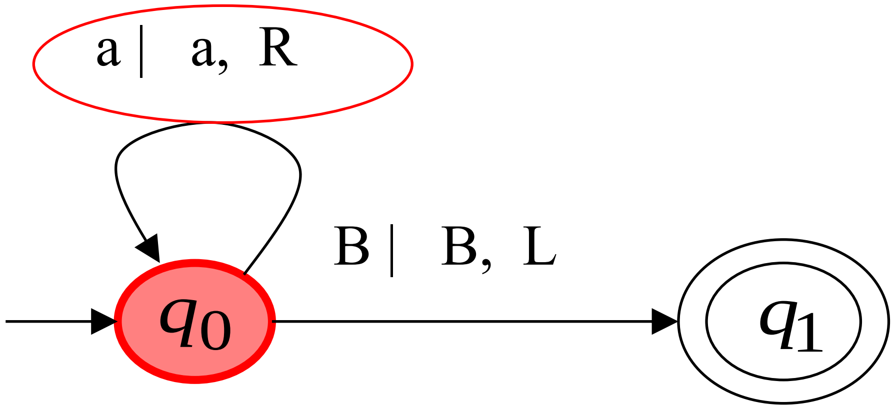




Turing Machine Example (cont..)

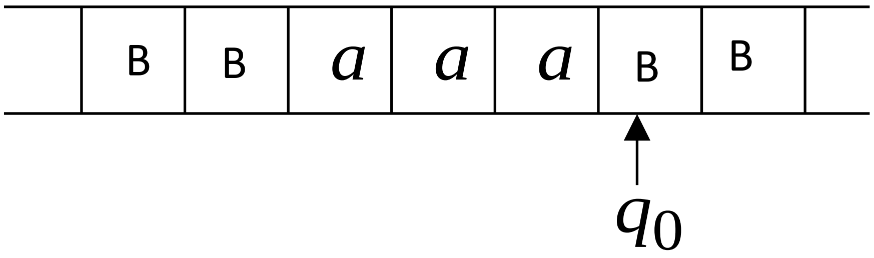
Time 2

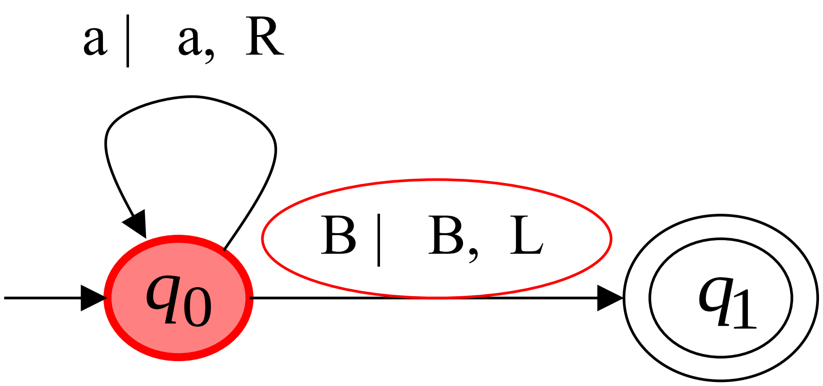




Turing Machine Example (cont..)

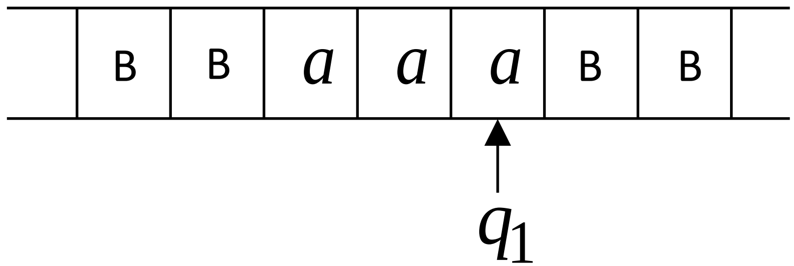
Time 3

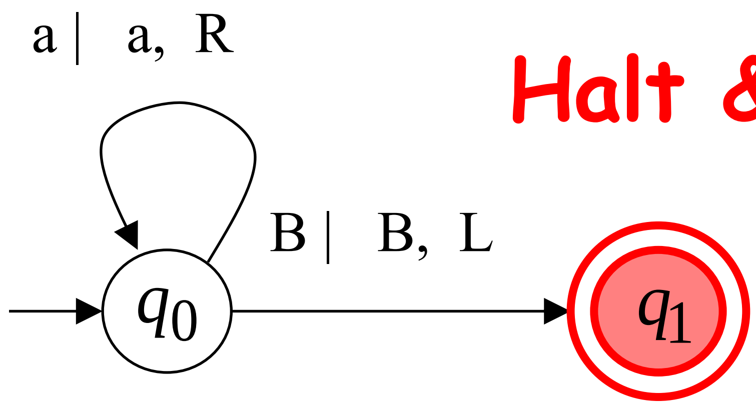




Turing Machine Example (cont..)

Time 4

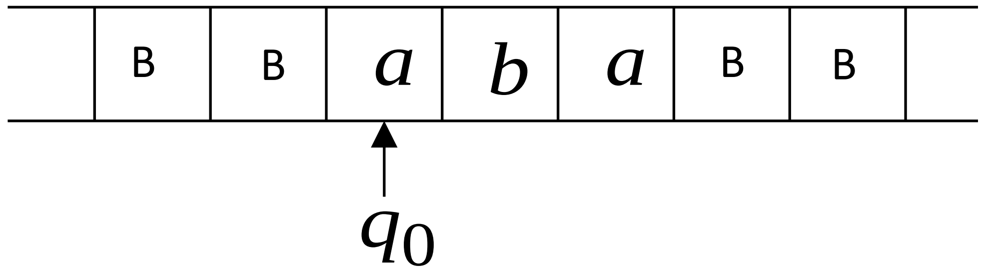


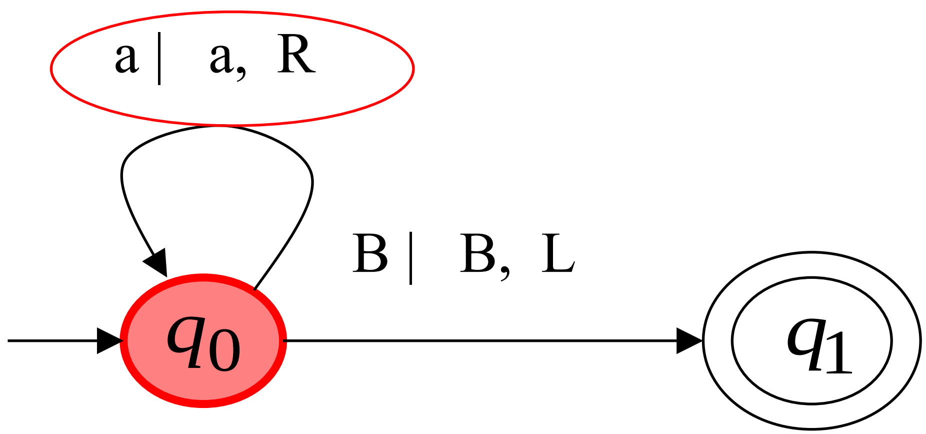


Halt & Accept

Rejection Example

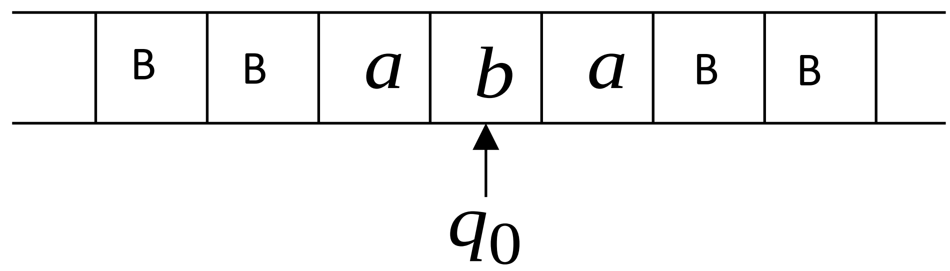
Time 0





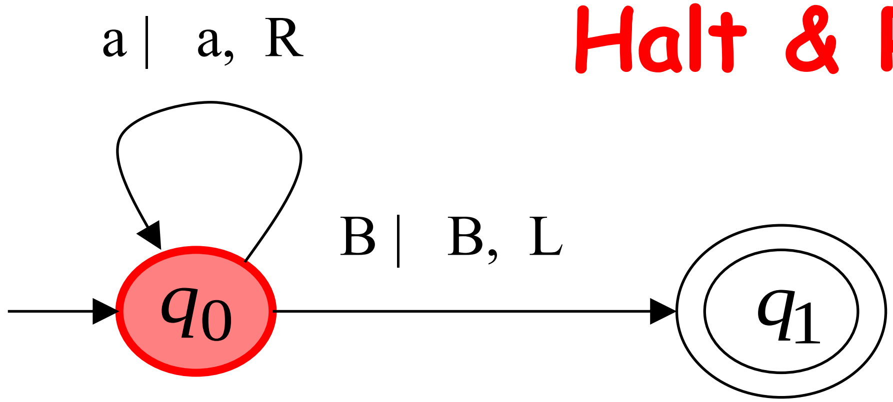
Rejection Example (cont..)

Time 1



No possible Transition

Halt & Reject



TM recognizable Problems

- A TM *recognizes* a language iff it accepts all and only those strings in the language.
- A language L is called Turing-recognizable or recursively enumerable iff some TM recognizes L.
- A TM *decides* a language L iff it accepts all strings in L and rejects all strings not in L.
- A language L is called decidable or recursive iff some TM decides L.

Example 1:

Q. Design Turing Machine to recognize L= {$a^n b^n$ | n≥ 1 }

Language : { ab, aabb, aaabbb, ...}

Logic : Replace input 'a', by ‘X‘ and move right till we get symbol ‘b’.

Replace input ‘b', by ‘Y‘ and move left till we get ‘X’.

Repeat for complete input string

$\Sigma = \{ a, b \}$

$\Gamma = \{ a, b, X, Y, B \}$

States : q0 : Read ‘a’ make it ‘X’ move right

q1 : Read ‘b’ make it ‘Y’ move left

q2 : Read ‘X’ keep it as ‘X’ move right

q3 : Check any ‘b’ is remaining

qf : Final state

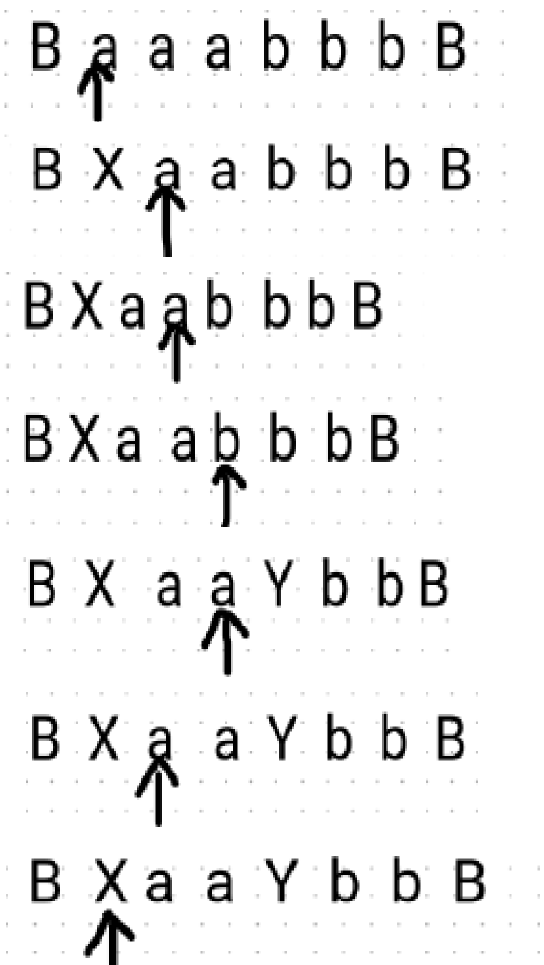
Initial state : q_0

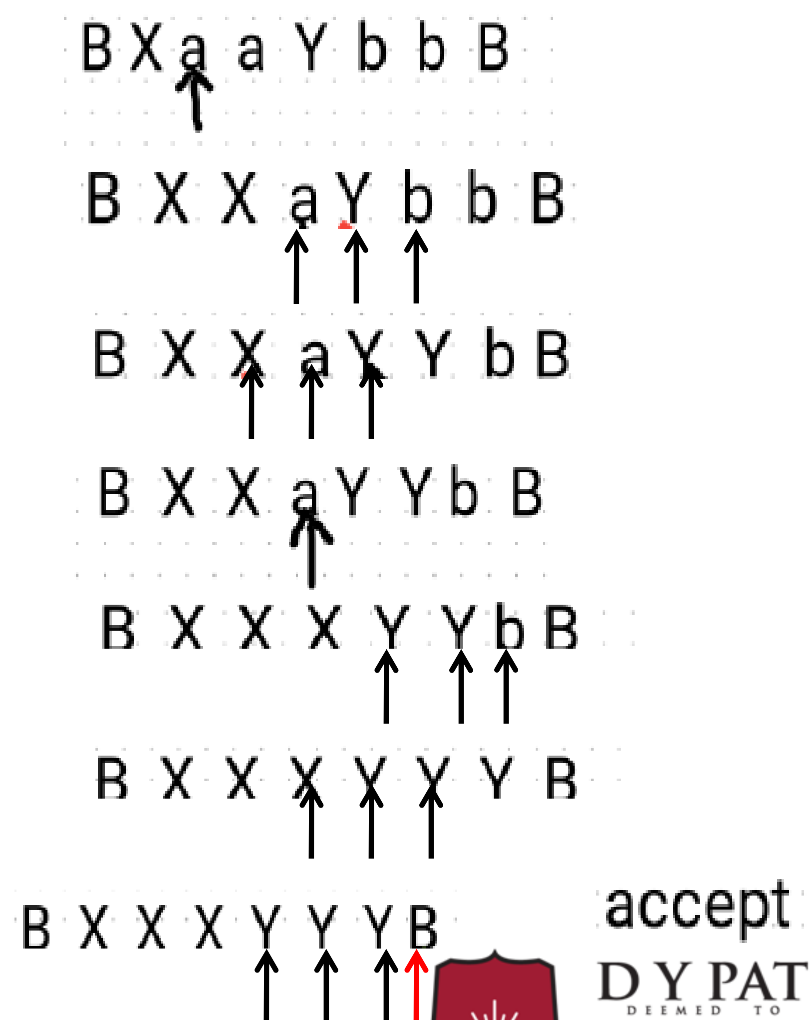
Final state : q_f

Example Processing

- Replace input 'a', by ‘X‘ and move right till we get ‘b’.
- Replace input ‘b', by ‘Y‘ and move left till we get ‘X’.
- Repeat for complete input string

Consider Input String = aaabbb





D Y PATIL
DEEMED TO BE UNIVERSITY
RAMRAO ADIK
INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

Logic in Detail

q0 – Replace ‘a’ by ‘X’ and move to right

q1 – Search for ‘b’ and replace by ‘Y’ and move left, skip all a’s and Y’s

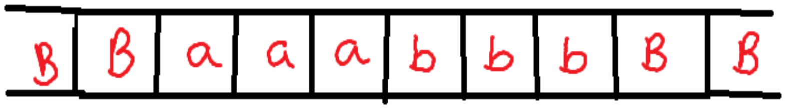
q2 – Search for ‘X’ and keep it as it is. Then move right and go to q0 state to repeat the cycle for all a’s and b’s. While doing this skip all a’s and Y’s.

q3 – On q0 state after moving right if we get ‘Y’ that means all a’s are over. Now, move right till we get blank symbol to check if any ‘b’ is remaining or not. Skip all Y’s

qf – After all Y’s if we get blank symbol that means there is no ‘b’ remaining so move to this final state.

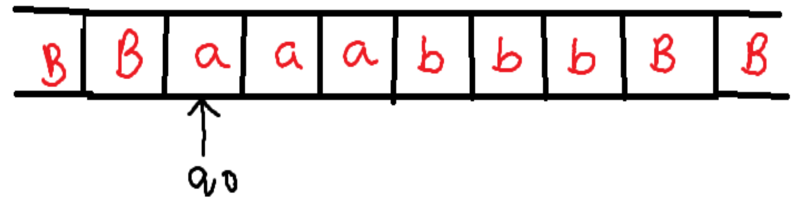
Example Processing with State Transitions

1



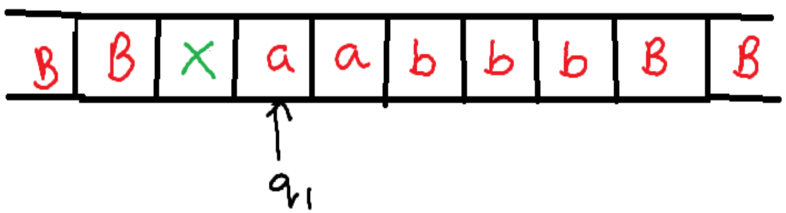
Input String: aaabbb

2



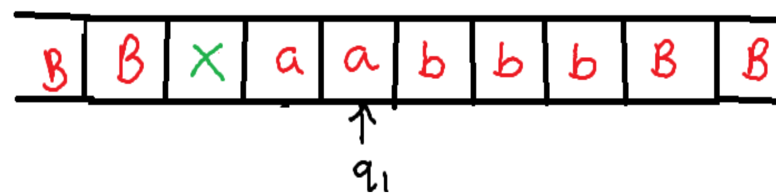
δ(q0, a) → { q1, X, R }

3



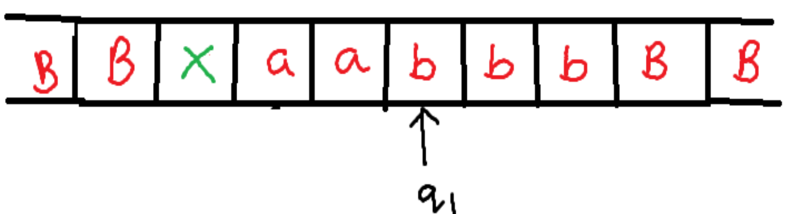
δ(q1, a) → { q1, a, R }

4



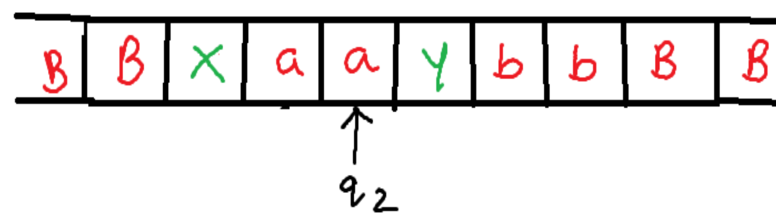
δ(q1, a) → { q1, a, R }

5



δ(q1, b) → { q2, Y, L }

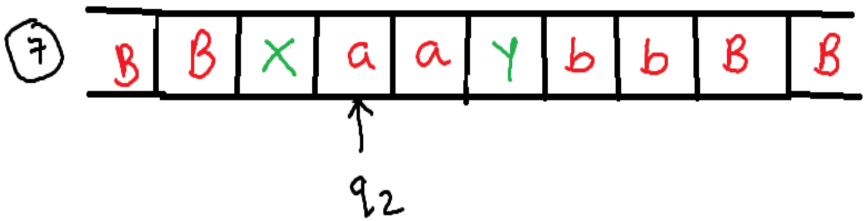
6



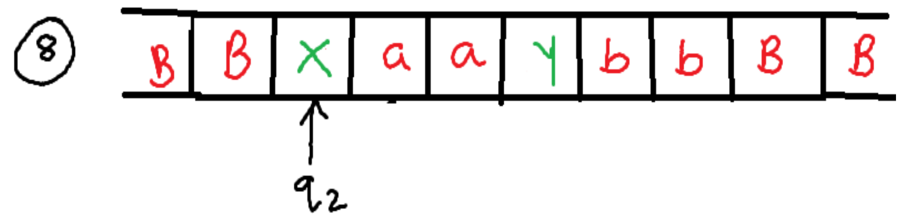
δ(q2, a) → { q2, a, L }

D Y PATIL DEEMED TO BE UNIVERSITY
RAMRAO ADIK INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

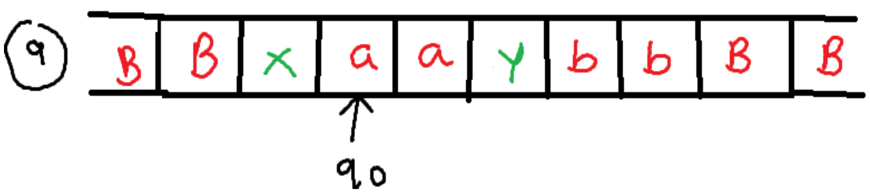
Example Processing with State Transitions (cont..)



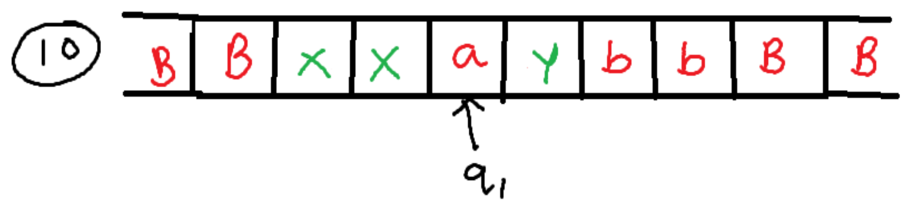
$\delta(q2, a) \rightarrow \{q2, a, L\}$



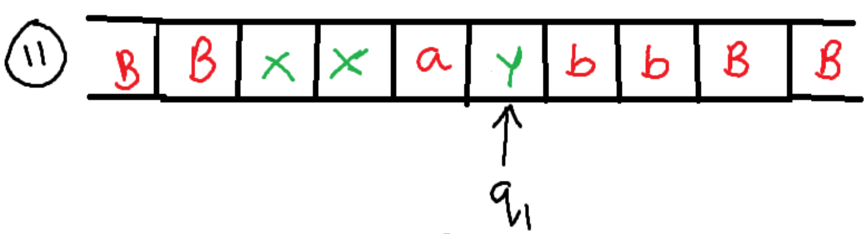
$\delta(q2, X) \rightarrow \{q0, X, R\}$



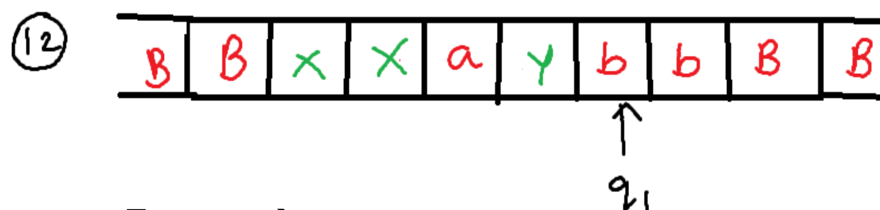
$\delta(q0, a) \rightarrow \{q1, X, R\}$



$\delta(q1, a) \rightarrow \{q1, a, R\}$

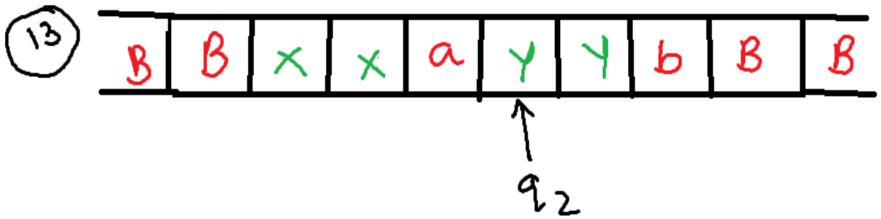


$\delta(q1, Y) \rightarrow \{q1, Y, R\}$

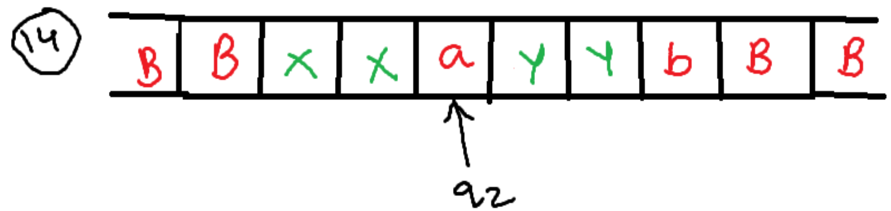


$\delta(q1, b) \rightarrow \{q2, Y, L\}$

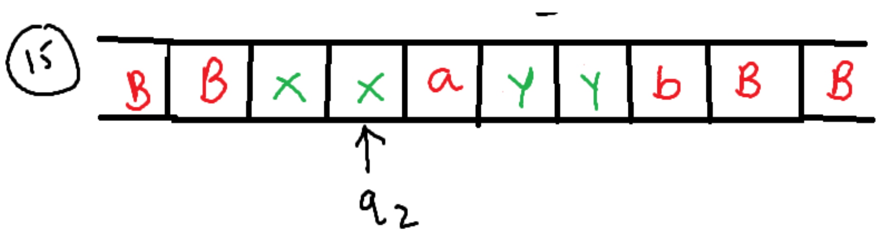
Example Processing with State Transitions (cont..)



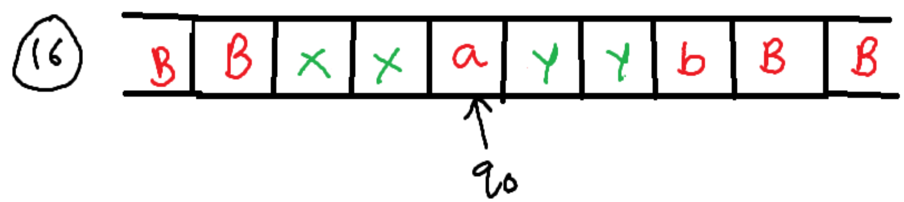
δ(q2, Y) → {q2, Y, L}



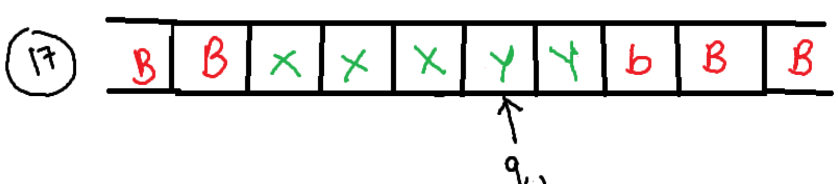
δ(q2, a) → {q2, a, L}



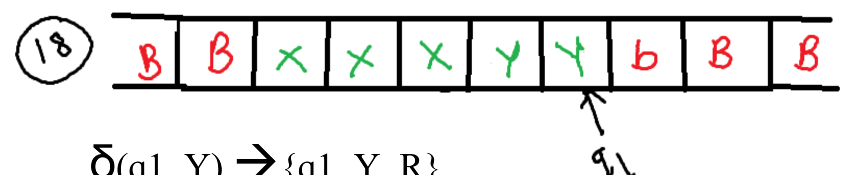
δ(q2, X) → {q0, X, R}



δ(q0, a) → {q1, X, R}

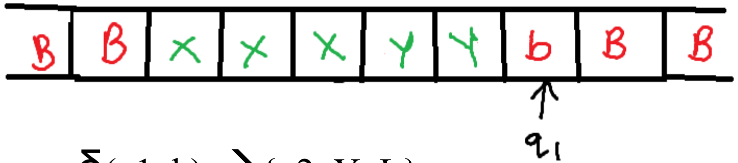


δ(q1, Y) → {q1, Y, R}



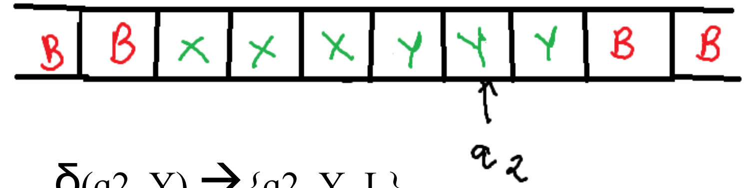
δ(q1, Y) → {q1, Y, R}

19



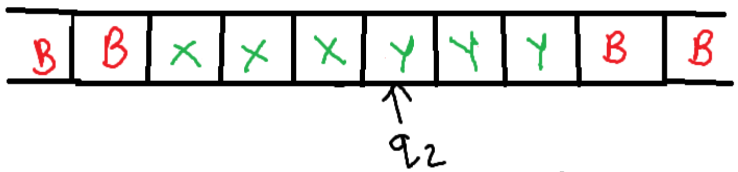
$\delta(q1, b) \rightarrow \{q2, Y, L\}$

20



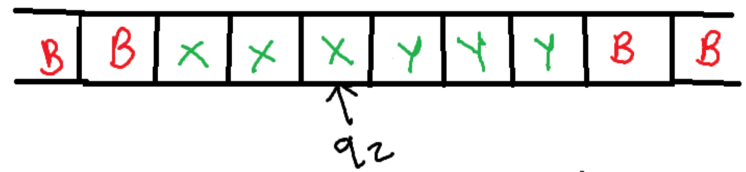
$\delta(q2, Y) \rightarrow \{q2, Y, L\}$

21



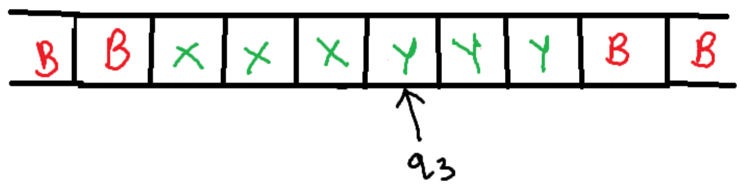
$\delta(q2, Y) \rightarrow \{q2, Y, L\}$

22



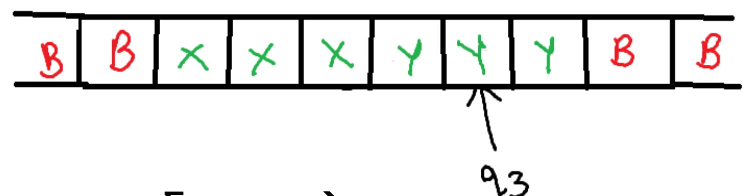
$\delta(q2, X) \rightarrow \{q0, X, R\}$

23



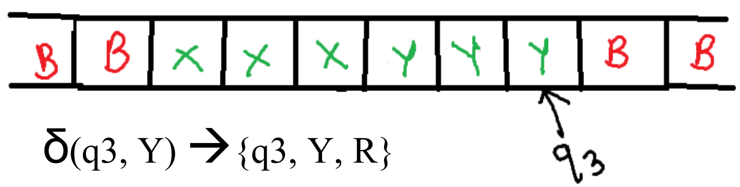
$\delta(q0, Y) \rightarrow \{q3, Y, R\}$

24



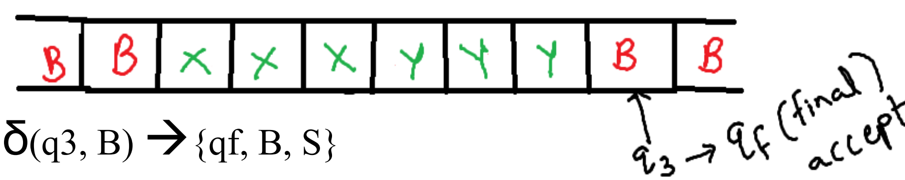
$\delta(q3, Y) \rightarrow \{q3, Y, R\}$

25



$\delta(q3, Y) \rightarrow \{q3, Y, R\}$

26

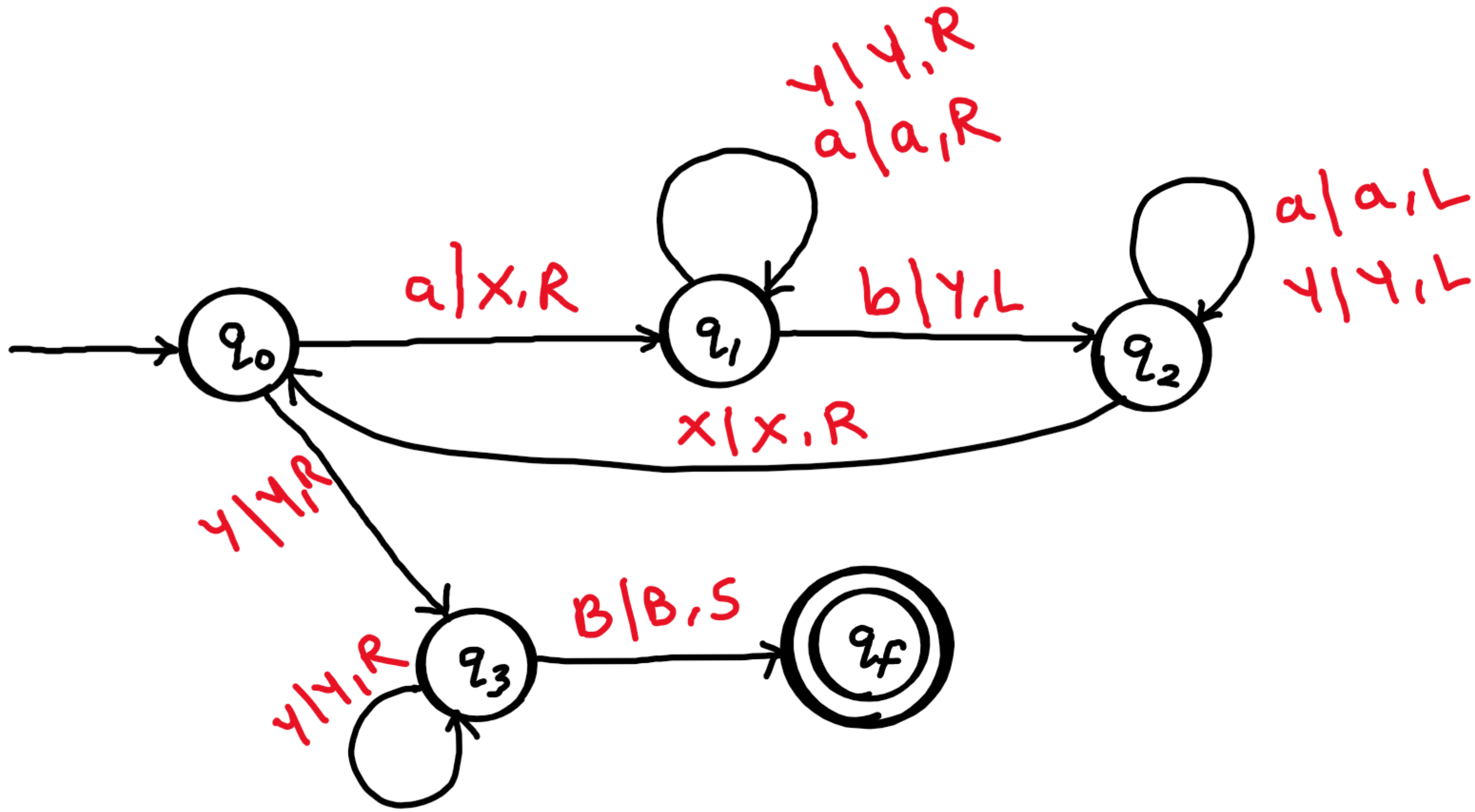


$\delta(q3, B) \rightarrow \{qf, B, S\}$

Transition Table

Q \ Γ	a	b	X	Y	B
q0	(q1, X, R)			(q3, Y, R)	
q1	(q1, a, R)	(q2, Y, L)		(q1, Y, R)	
q2	(q2, a, L)		(q0, X, R)	(q2, Y, L)	
q3				(q3, Y, R)	(qf, B, S)
qf*	Final State				

Transition Diagram



D Y PATIL
DEEMED TO BE
UNIVERSITY
RAMRAO ADIK
INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

Thank You

Lecture No 44:
Turing Machine Examples

Example 2:

Q. Design Turing Machine to recognize L= {$a^n b^{n+1}$ | n≥ 1 }

Language : { abb, aabbb, aaabbbb, ...}

Logic :

Replace input 'a', by 'X' and move right till we get symbol 'b'.

Replace input 'b', by 'Y' and move left till we get 'X'.

Repeat till all a's are over

When a's are over search for last 'b'

$\sum = \{ a, b \}$

$\Gamma = \{ a, b, X, Y, B \}$

States :

q0 : Read 'a' make it 'X' move right

q1 : Read 'b' make it 'Y' move left

q2 : Search 'X' keep it as 'X' move right

q3 : Search for last 'b'

q4 : Extra 'b'

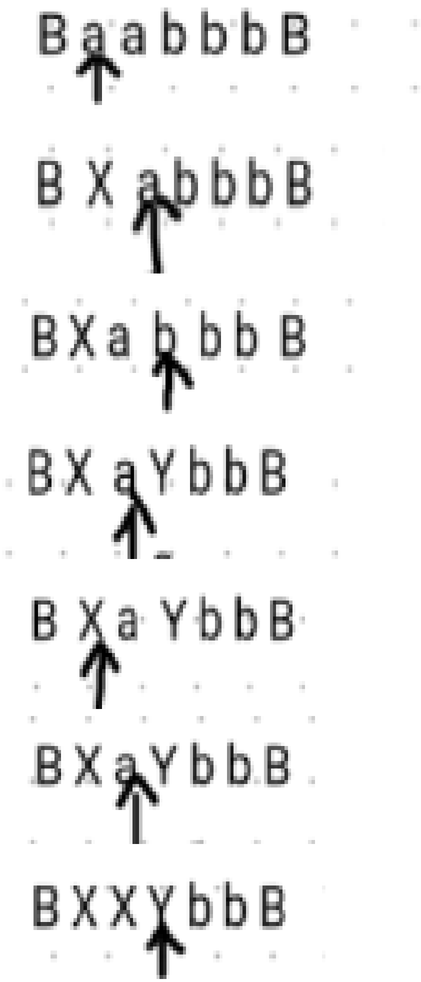
qf : Final state

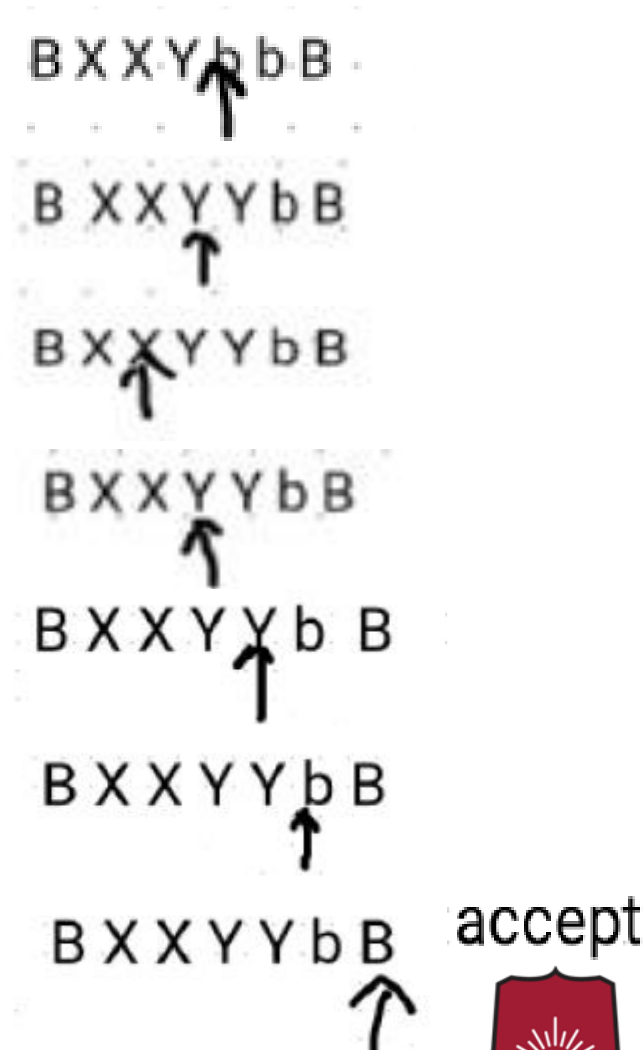
Initial state : q_0 **Final state :** q_f

Example Processing

- Replace input 'a', by ‘X‘ and move right till we get ‘b’.
- Replace input ‘b', by ‘Y‘ and move left till we get ‘X’.
- Repeat till all a’s are over
- When a’s are over search for last ‘b’

Consider Input String = aabbb





Logic in Detail

q0 – Replace ‘a’ by ‘X’ and move to right

q1 – Search for ‘b’ and replace by ‘Y’ and move left, skip all a’s and Y’s

q2 – Search for ‘X’ and keep it as it is and move right. Go to q0 state to repeat the cycle. While doing this skip all a’s and Y’s.

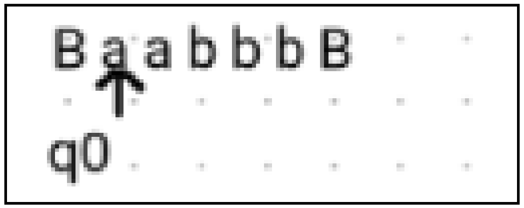
q3 – On q0 state after moving right if we get ‘Y’ that means all a’s are over. Now, move right to search for the last ‘b’. Skip all Y’s.

q4 – After all Y’s if we get ‘b’, this indicates one extra ‘b’ than ‘a’ is found.

qf – After the last ‘b’ if we get blank symbol then move to final state. This ensures that there is only one extra ‘b’ present.

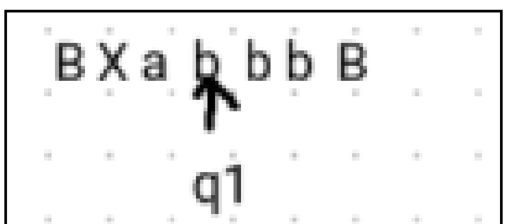
Example Processing with State Transitions

Input String: aabbb

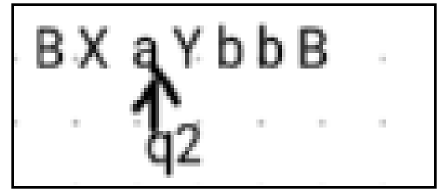


$\delta(q0, a) \rightarrow \{ q1, X, R \}$

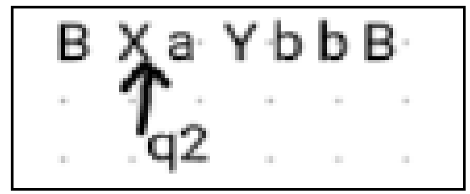
$\delta(q1, a) \rightarrow \{ q1, a, R \}$



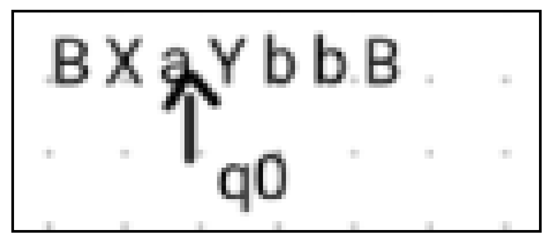
$\delta(q1, b) \rightarrow \{ q2, Y, L \}$



$\delta(q2, a) \rightarrow \{ q2, a, L \}$

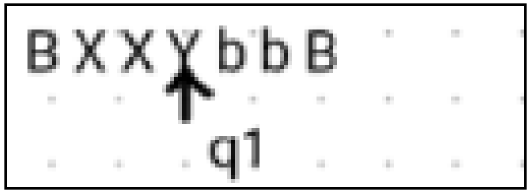


$\delta(q2, X) \rightarrow \{q0, X, R\}$

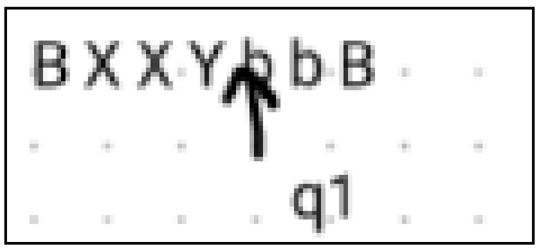


$\delta(q0, a) \rightarrow \{q1, X, R\}$

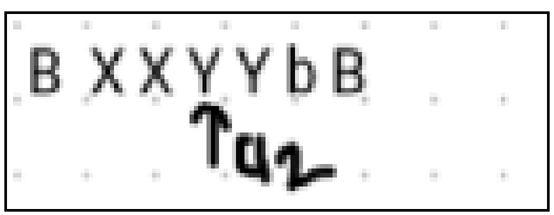
Example Processing with State Transitions (cont..)



δ(q1, Y) → {q1, Y, R}



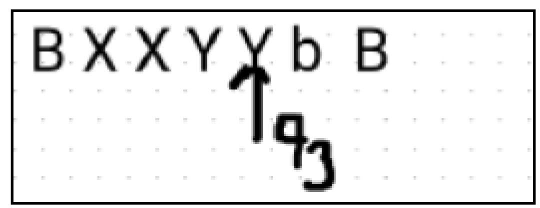
δ(q1, b) → {q2, Y, L}



δ(q2, Y) → { q2, Y, L }

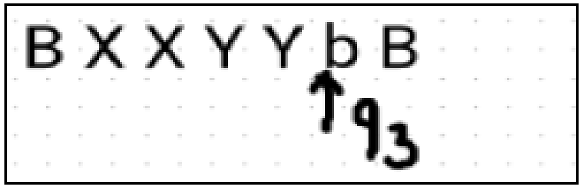
δ(q2, X) → {q0, X, R}

δ(q0, Y) → {q3, Y, R}

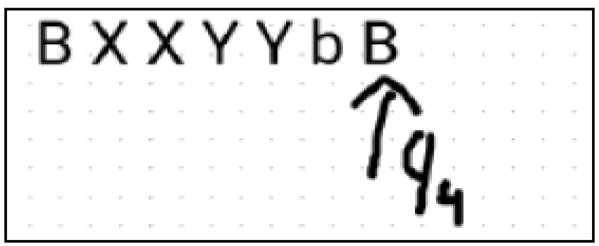


δ(q3, Y) → {q3, Y, R}

Example Processing with State Transitions (cont..)



δ(q3, b) → {q4, b, R}



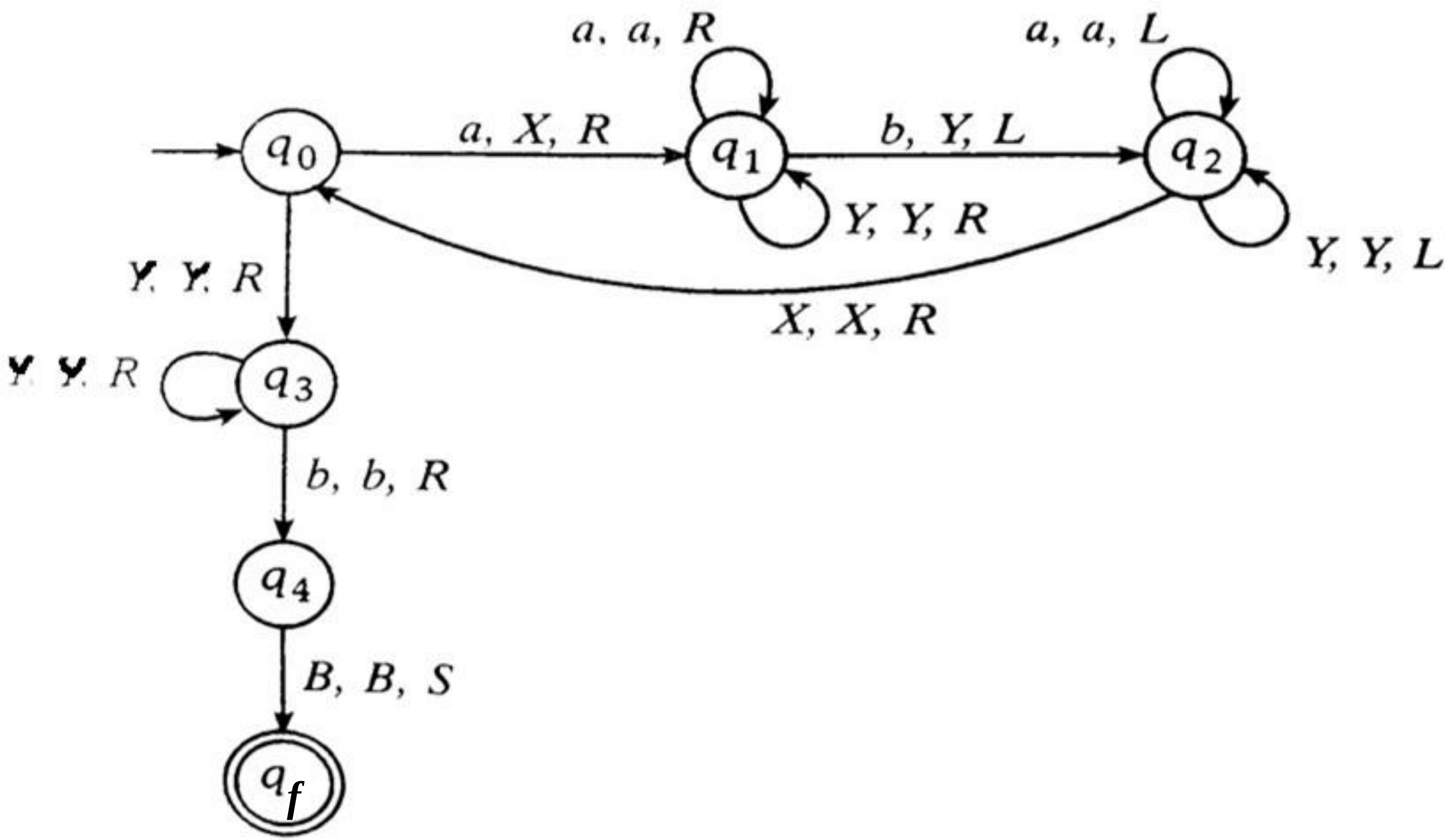
δ(q4, B) → {qf, B, S}

String accepted

Transition Table

Q \ Γ	a	b	X	Y	B
q0	(q1, X, R)			(q3, Y, R)	
q1	(q1, a, R)	(q2, Y, L)		(q1, Y, R)	
q2	(q2, a, L)		(q0, X, R)	(q2, Y, L)	
q3		(q4, b, R)		(q3, Y, R)	
q4					(qf, B, S)
qf*	Final State				

Transition Diagram



D Y PATIL
DEEMED TO BE
UNIVERSITY
RAMRAO ADIK
INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

Thank You