Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

Theory of Computation

SY BTech

Unit V

Turing Machine

Course Objective & Course Outcomes

- **Course Objectives:**

  1. To Study computing machines by describing, classifying and comparing different types of computational models.
  2. Encourage students to study Theory of Computability and Complexity.

- **Course Outcomes:**

  After successful completion of this course students will be able to:

1. Construct finite state machines to solve problems in computing
2. Write mathematical expressions for the formal languages
3. Apply well defined rules for syntax verification
4. Construct and analyze Push Down, Post and Turing Machine for formal languages
5. Express the understanding of the decidability and Undecidability problems
6. Express the understanding of computational complexity.

Text Books & Reference Books

- **Text Books**

1. Michael Sipser “Introduction to the Theory of Computation” CENGAGE Learning, 3rd Edition ISBN-13:978-81-315-2529-6
2. Vivek Kulkarni, “Theory of Computation”, Oxford University Press, ISBN-13: 978-0-19-808458-7

- **Reference Books**

1. Hopcroft Ulman, “Introduction To Automata Theory, Languages And Computations”, Pearson Education Asia, 2nd Edition
2. Daniel. A. Cohen, “Introduction to Computer Theory” Wiley-India, ISBN:978-81-265-1334-5
3. K.L.P Mishra ,N. Chandrasekaran ,“Theory Of Computer Science (Automata, Languages and Computation)”, Prentice Hall India,2nd Edition
4. John C. Martin, “Introduction to Language and Theory of Computation”, TMH, 3rd Edition ISBN: 978-0-07-066048-9
5. Kavi Mahesh, “Theory of Computation: A Problem Solving Approach”, Wiley-India, ISBN: 978-81-265-3311-4

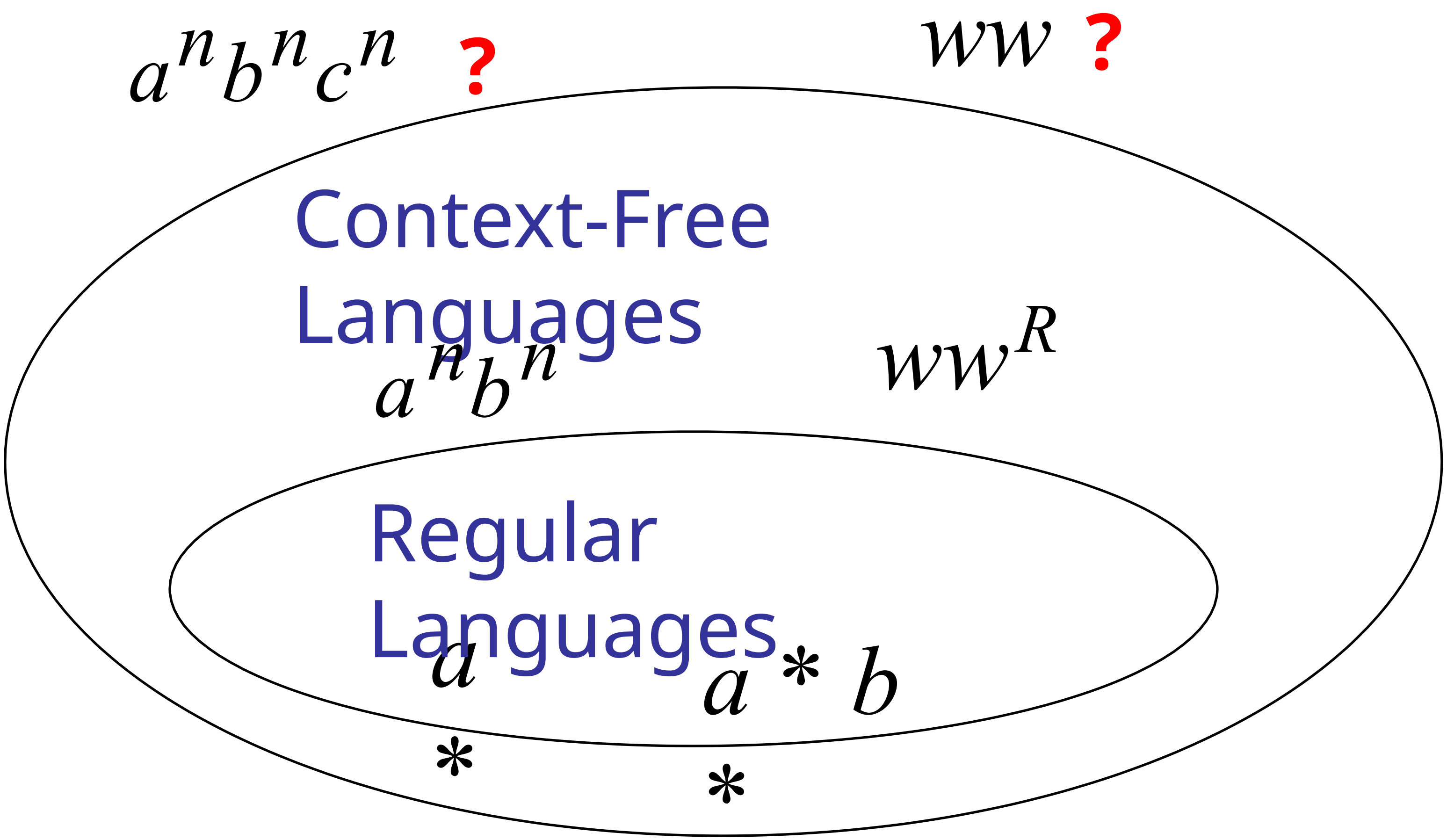
Church-Turing's Thesis

Everything that is algorithmically computable is computable by a Turing machine

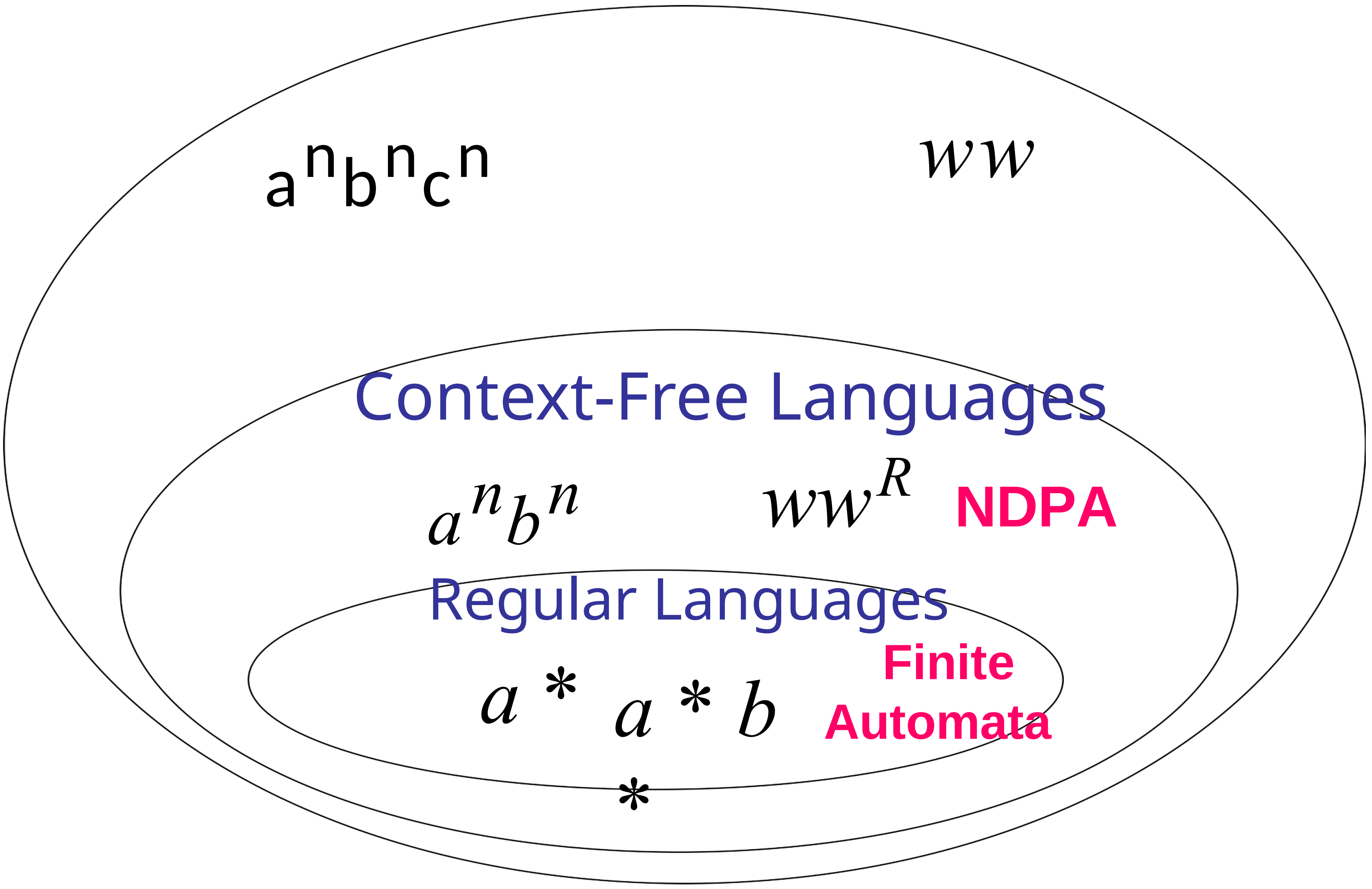
Turing Machine

- Invented by Alan Turing in 1936
- A simple mathematical model of a general purpose computer
- It is capable of performing any calculation which can be performed by any computing machine

The Language Hierarchy

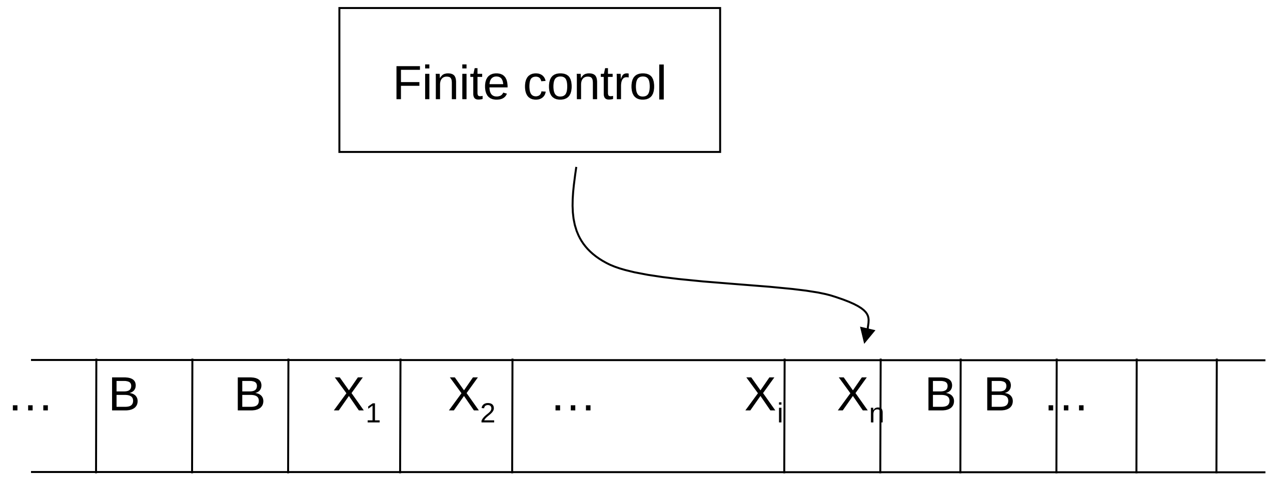


Language Accepted by TM



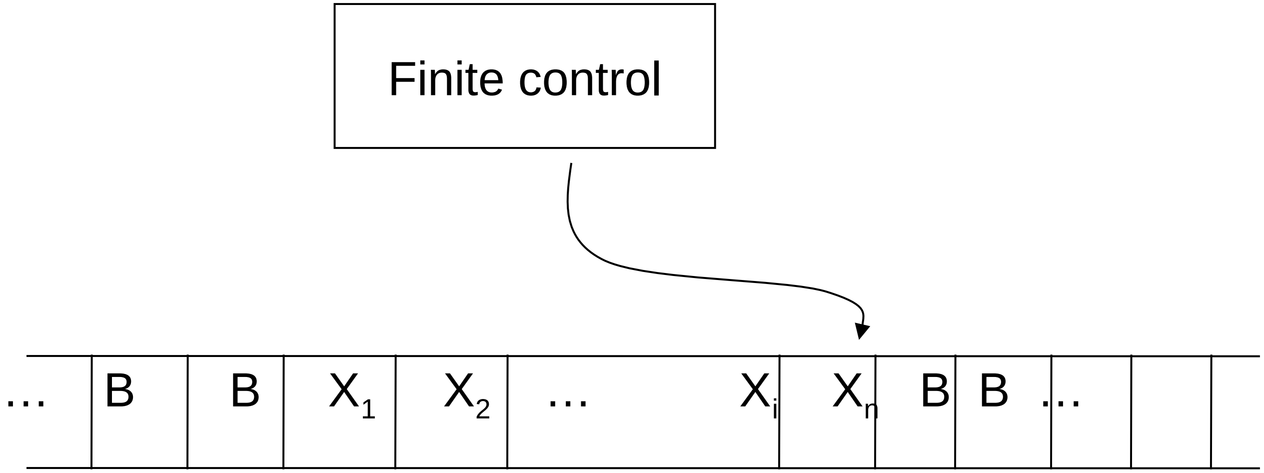
Elements of a Turing Machine

- A TM consists of the following:
  - A tape head : read / write a symbol at a time, move to left/right
  - An infinite tape: contains square cells in which symbols can be written
  - A finite set of symbols
  - A finite set of states



Turing Machine

- A TM consists of a finite control (i.e. a finite state automaton) that is connected to an infinite tape.
- Initially the input consists of a finite-length string of symbols and is placed on the tape.
- To the left of the input and to the right of the input, extending to infinity, are placed blanks.
- The tape head is initially positioned at the leftmost cell holding the input.



Turing Machine

- In one move the TM will:
  - Change state, which may be the same as the current state
  - Write a tape symbol in the current cell, which may be the same as the current symbol
  - Move the tape head left or right one cell

Formal Definition of TM

Formally, the Turing Machine is denoted by the 7-tuple:

$$\mathbf{M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)}$$

Q = finite states of the control
Σ = finite set of input symbols, which is a subset of Γ below
Γ = finite set of tape symbols

δ = transition function. **$\delta : (Q \times \Gamma) \rightarrow (Q \times \Gamma \times D)$**

e.g. $\delta(q, X) = (p, Y, \{L, R, N\})$

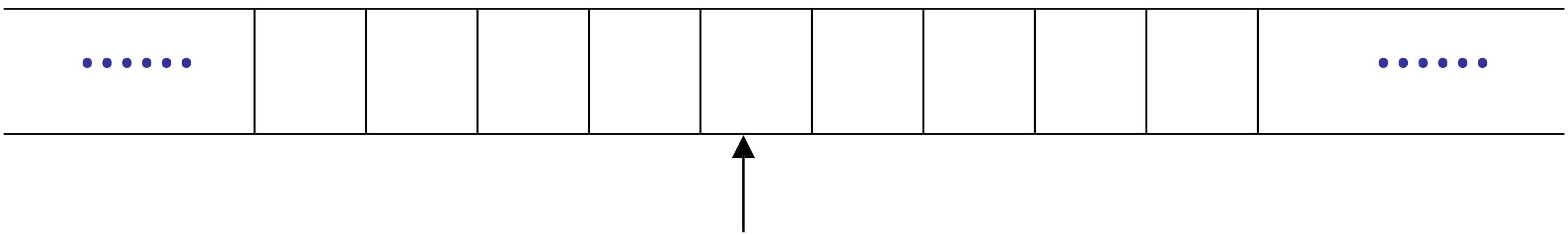
Where p = next state, Y = new symbol written on the tape,

D = direction to move the tape head(left, right, No move)

q_0= start state for finite control
B = blank symbol. **This symbol is in Γ but not in Σ.**
F = set of final or accepting states of Q.

The Tape

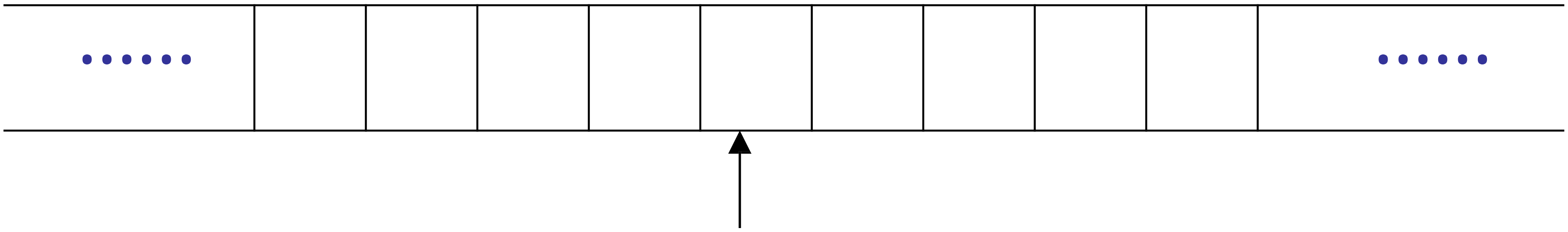
No boundaries -- infinite length



Read-Write head

The head moves Left or Right

The Tape



Read-Write head

The head at each time step:

1. Reads a symbol
2. Writes a symbol
3. Moves Left or Right

Transition Function

Takes two arguments:

1. A state, in Q
2. A tape symbol in Γ

δ(q, Z) is either undefined or a triple of the form (p, Y, D) where

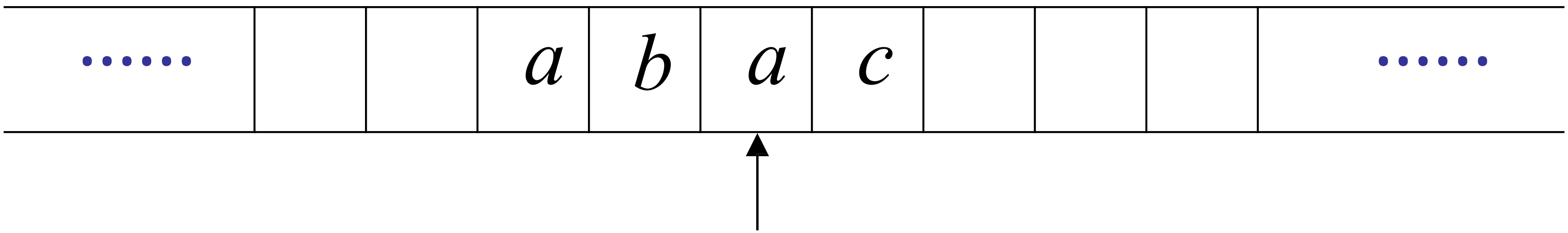
p is a state

Y is the new tape symbol

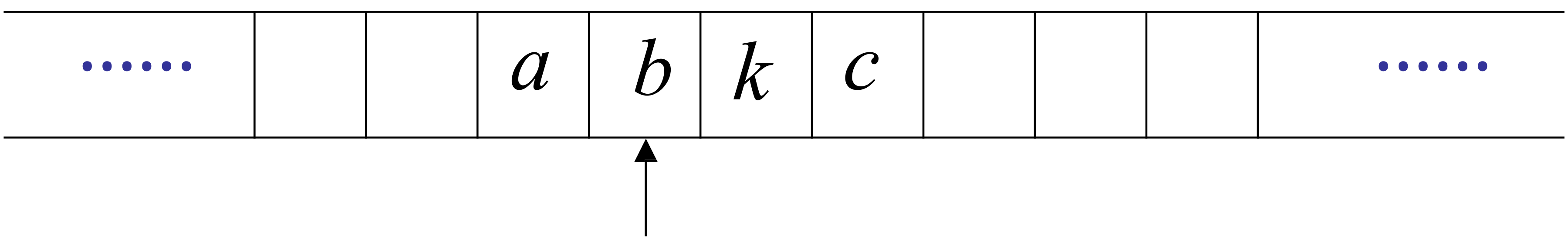
D is a direction, L or R

Example:

Time 0



Time 1

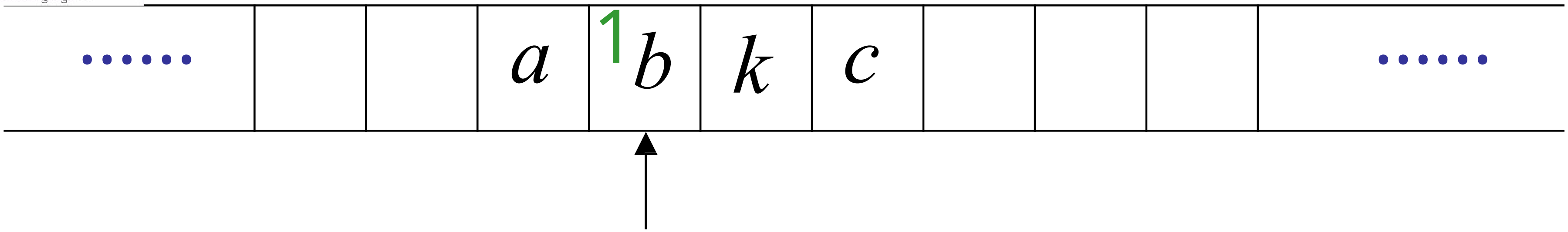


1. Reads a

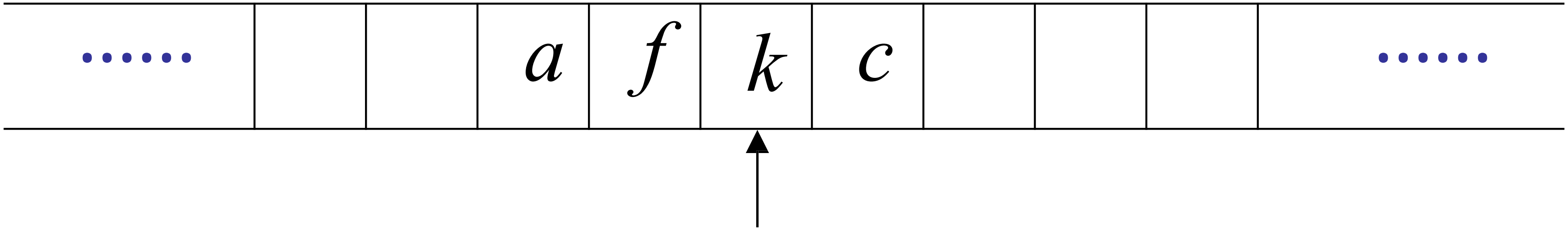
2. Writes k

3. Moves Left

Time 1



Time 2

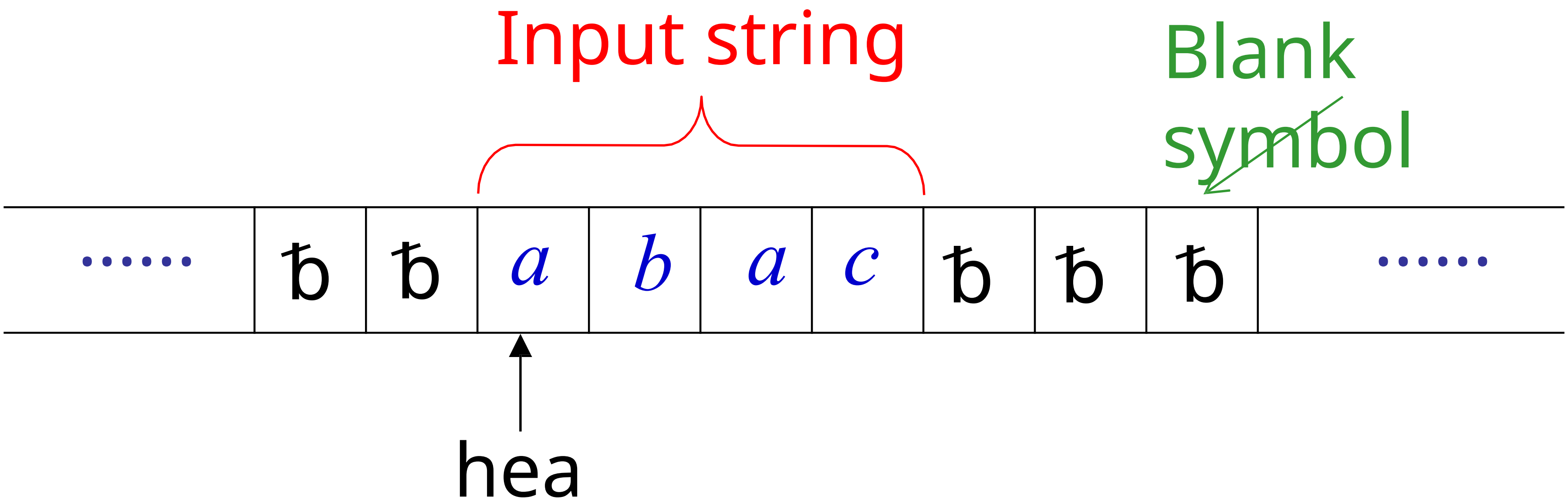


1. Reads b

2. Writes f

3. Moves Right

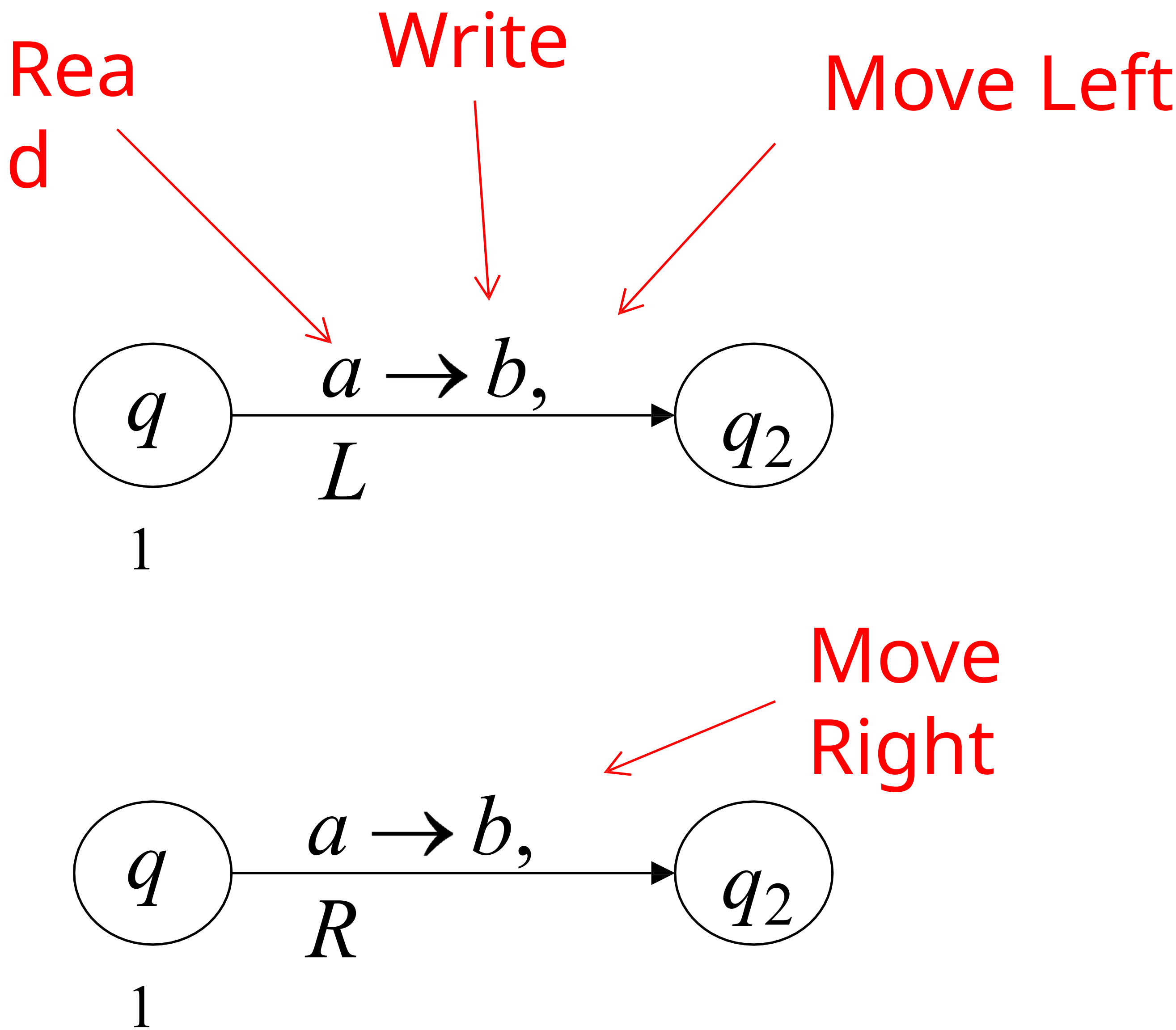
The Input String

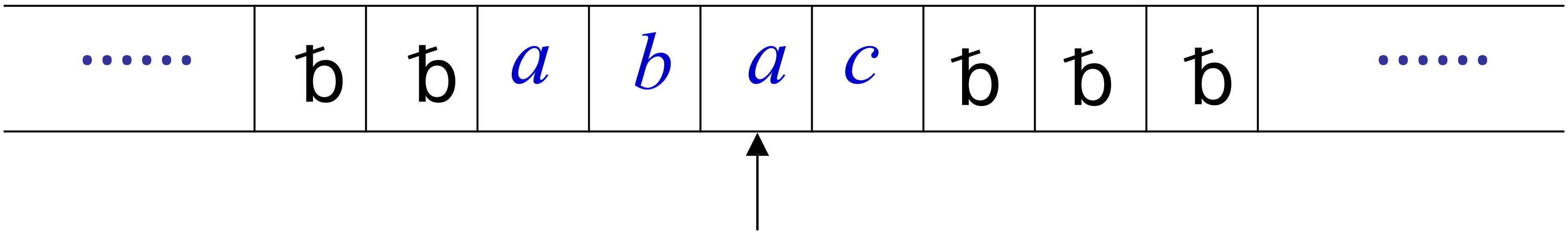


Head starts at the leftmost position of the input string

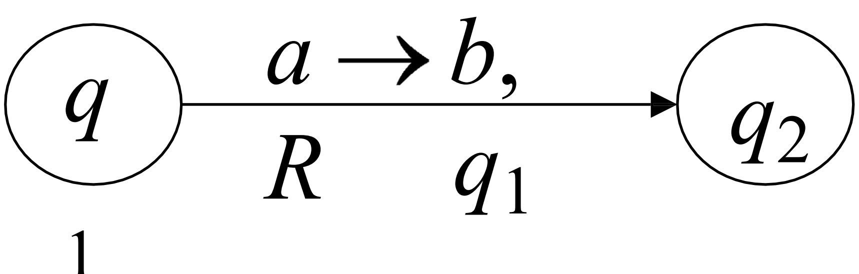
‘ƀ’ **are treated as left and right brackets for the input written on the tape.**

States & Transitions





Example:



current

Time 1

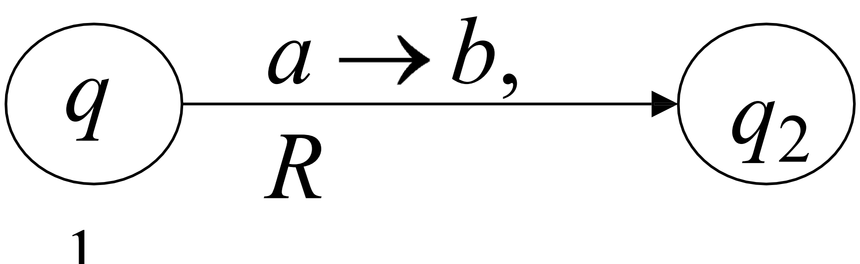
……	ƀ	ƀ	a	b	a	c	ƀ	ƀ	ƀ	……

q_1

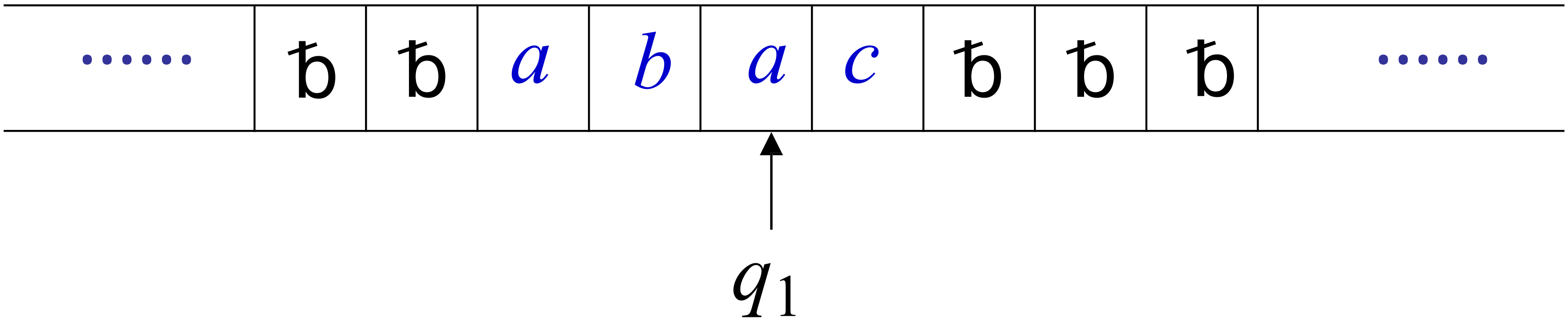
Time 2

……	ƀ	ƀ	a	b	b	c	ƀ	ƀ	ƀ	……

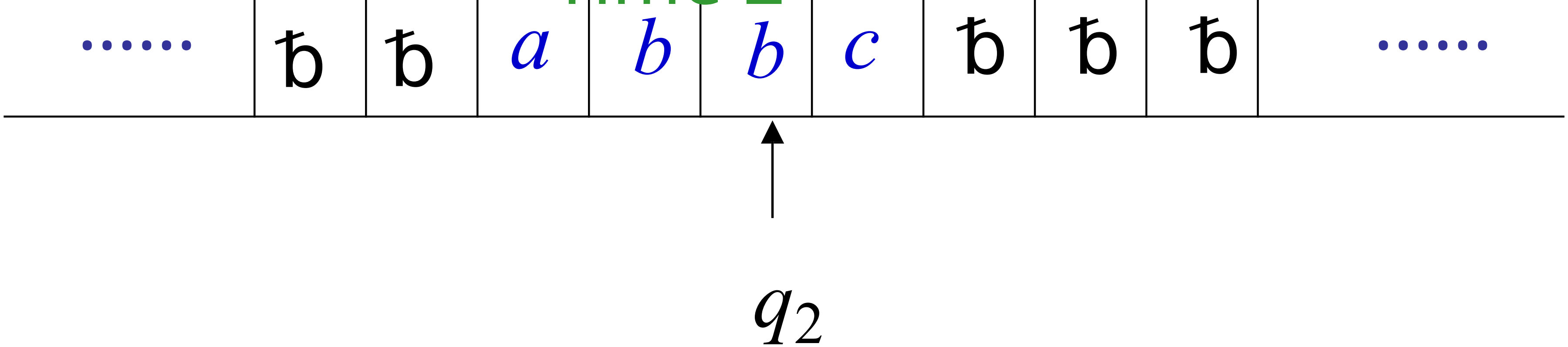
q_2

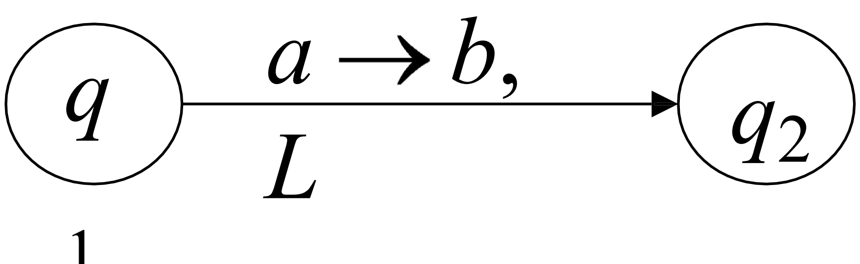


Time 1

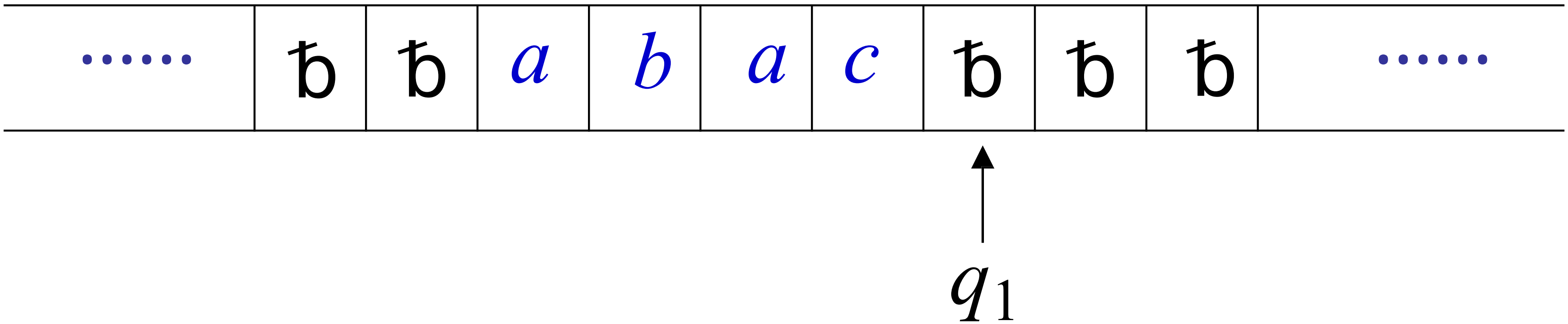


Time 2

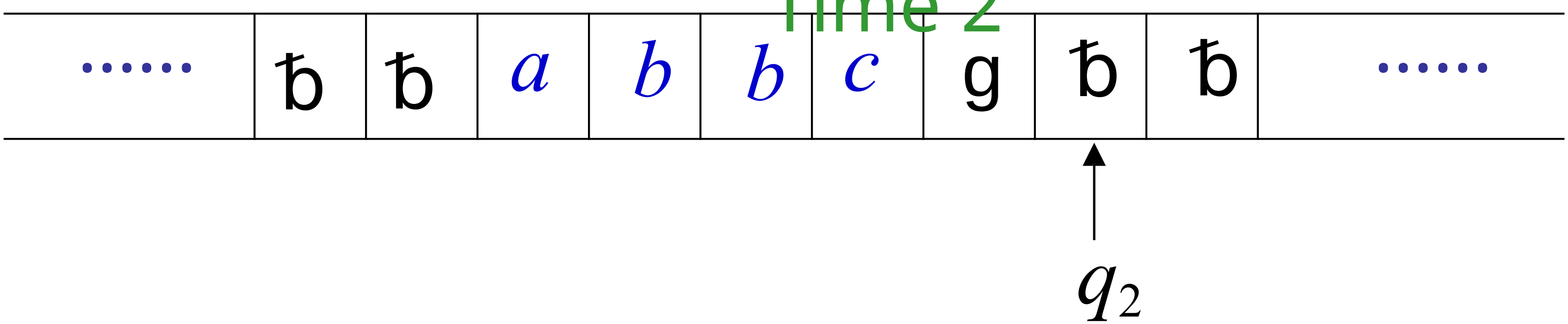


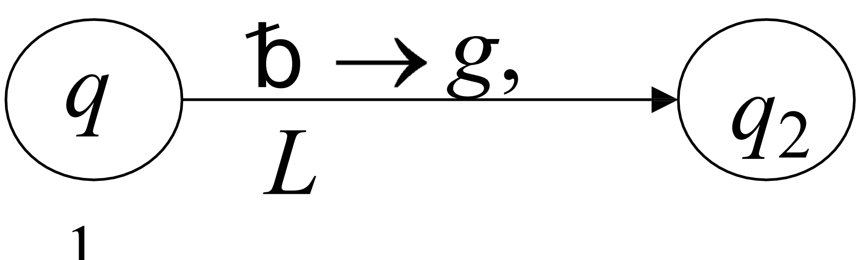


Time 1



Time 2



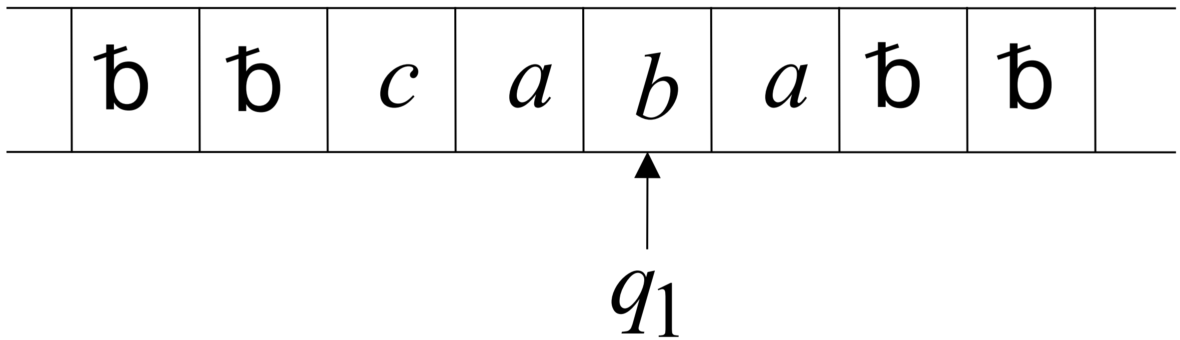


Instantaneous Descriptions(ID) of a Turing Machine

- Sometimes it is useful to describe what a TM does in terms of its ID (instantaneous description), just as we did with the PDA

- The ID shows all non-blank cells in the tape, pointer to the cell the head is over with the name of the current state
  - use the turnstile symbol $\vdash$ to denote the move.
  - As before, to denote zero or many moves, we can use $\vdash^*$

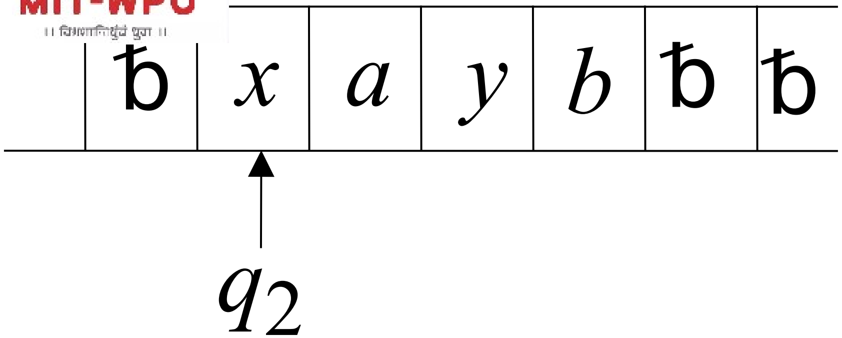
Instantaneous Descriptions(ID) of a TM

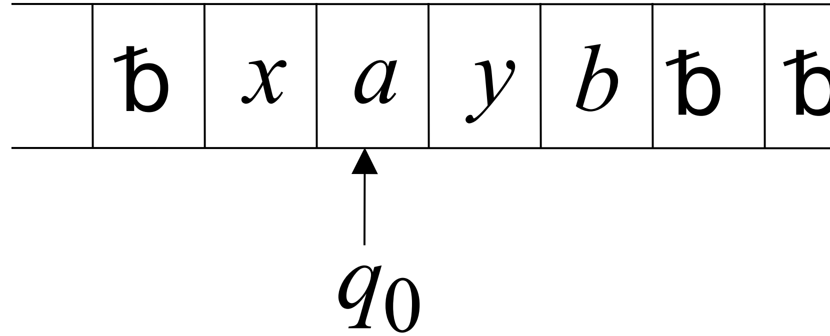


$$ca\ q_1\ ba$$

(1) For constructing the ID, we simply insert the current state in the input string to the left of the symbol under the R/W head.

(2) We observe that the blank symbol may occur as part of the left or right substring.





$$q_2\ xayb \succ x\ q_0\ ayb$$

Instantaneous Descriptions(ID) of a TM

Moves in a TM

As in the case of pushdown automata, $\delta(q, x)$ induces a change in ID of the Turing machine. We call this change in ID a move.

Suppose $\delta(q, x_i) = (p, y, L)$. The input string to be processed is $x_1x_2 \ldots x_n$, and the present symbol under the R/W head is x_i. So the ID before processing x_i is

$$x_1x_2 \ldots x_{i-1}qx_i \ldots x_n$$

After processing x_i, the resulting ID is

$$x_1 \ldots x_{i-2}\, px_{i-1}yx_{i+1} \ldots x_n$$

This change of ID is represented by

$$x_1x_2 \ldots x_{i-1}\, q\, x_i \ldots x_n \vdash x_i \ldots x_{i-2}\, px_{i-1}\, yx_{i+1} \ldots x_n$$

If $i = 1$, the resulting ID is $p\, y\, x_2x_3 \ldots x_n$.

If $\delta(q, x_i) = (p, y, R)$, then the change of ID is represented by

$$x_1x_2 \ldots x_{i-1}q\, x_i \ldots x_n \vdash x_1x_2 \ldots x_{i-1}\, y\, px_{i+1} \ldots x_n$$

Example

Present state	Tape symbol				
	0	1	x	y	b
$\rightarrow q_1$	xRq_2				bRq_5
q_2	$0Rq_2$	yLq_3		yRq_2	
q_3	$0Lq_4$		xRq_5	yLq_3	
q_4	$0Lq_4$		xRq_1		
q_5				$yxRq_5$	bRq_6
q_6					

(a) $q_1 011 \vdash xq_2 11 \vdash q_3 xy1 \vdash xq_5 y1 \vdash xyq_5 1$

(b) $q_10011 \vdash xq_2011 \vdash x0q_211 \vdash xq_30y1 \vdash q_4x0y1 \vdash xq_10y1$.

$$\vdash xxq_2y1 \vdash xxyq_21 \vdash xxq_3yy \vdash xq_3xyy \vdash xxq_5yy$$

$$\vdash xxyq_5y \vdash xxyyq_5b \vdash xxyybq_6$$

M halts. As q_6 is an accepting state, the input string 0011 is accepted by M.

(c) $q_1001 \vdash xq_201 \vdash x0q_21 \vdash xq_30y \vdash q_4x0y$

$$\vdash xq_10y \vdash xxq_2y \vdash xxyq_2$$

M halts. As q_2 is not an accepting state, 001 is not accepted by M.

Acceptance of Input

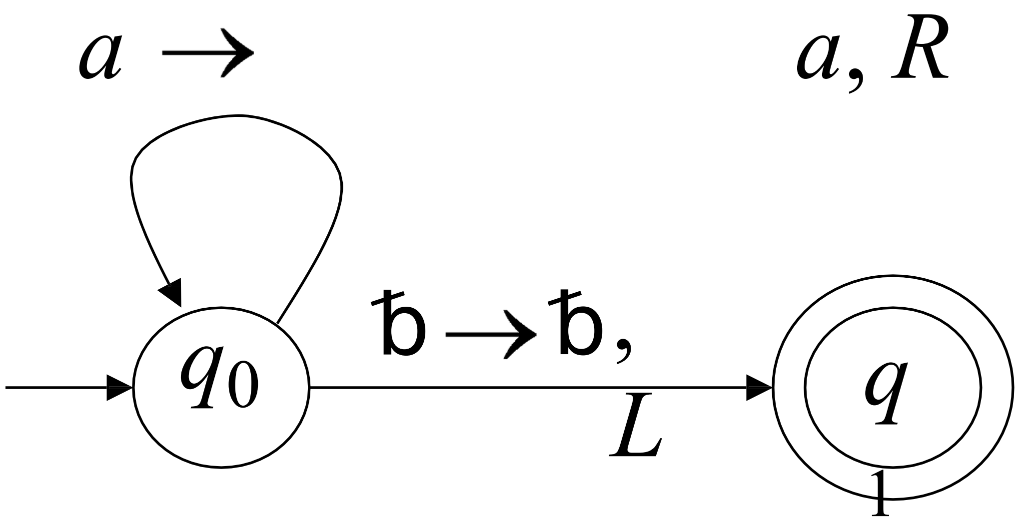
Accept Input → If machine halts in a final state

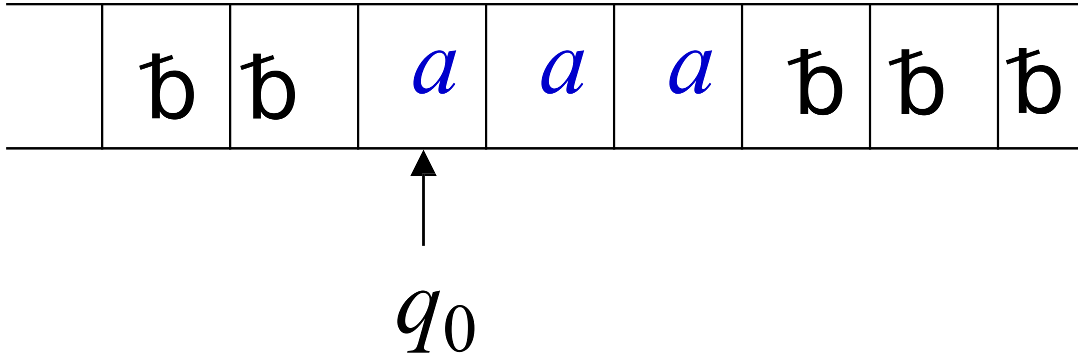
Reject Input → If machine halts in a non-final state

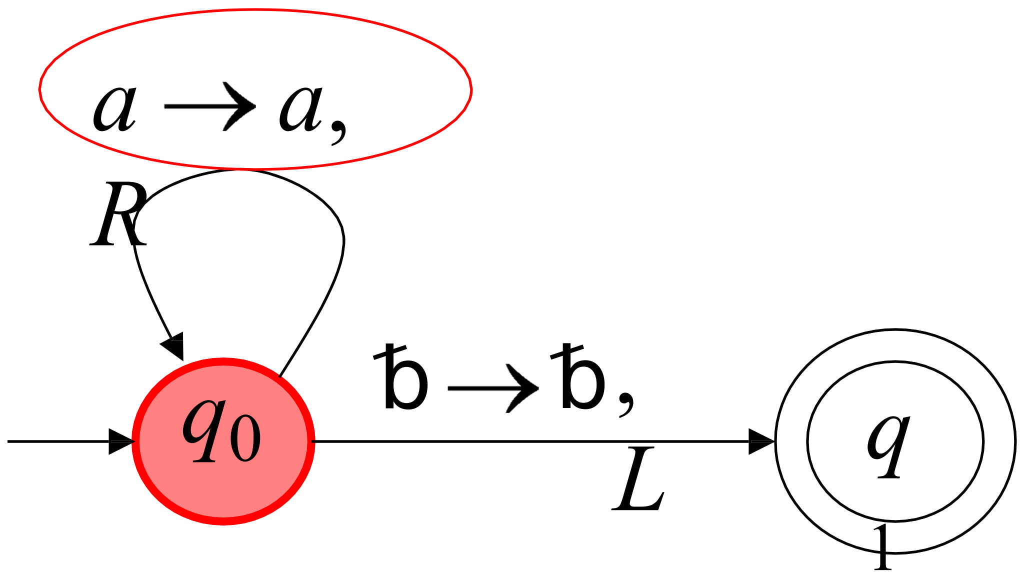
or

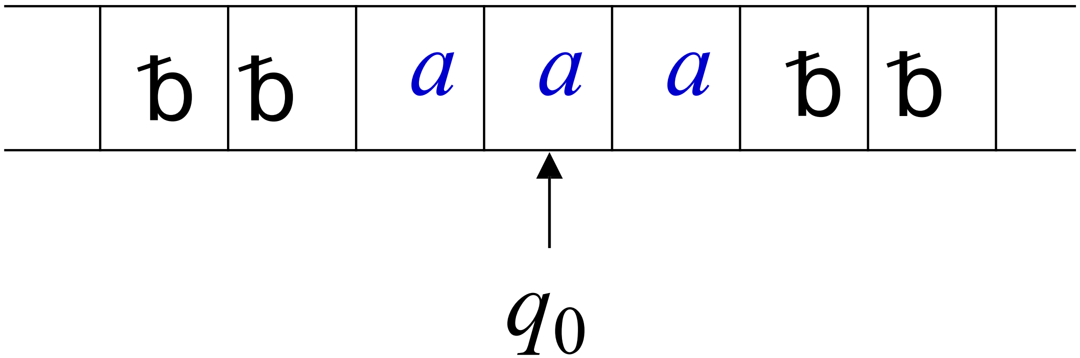
If machine enters an *infinite loop*

Example 1 :aa*

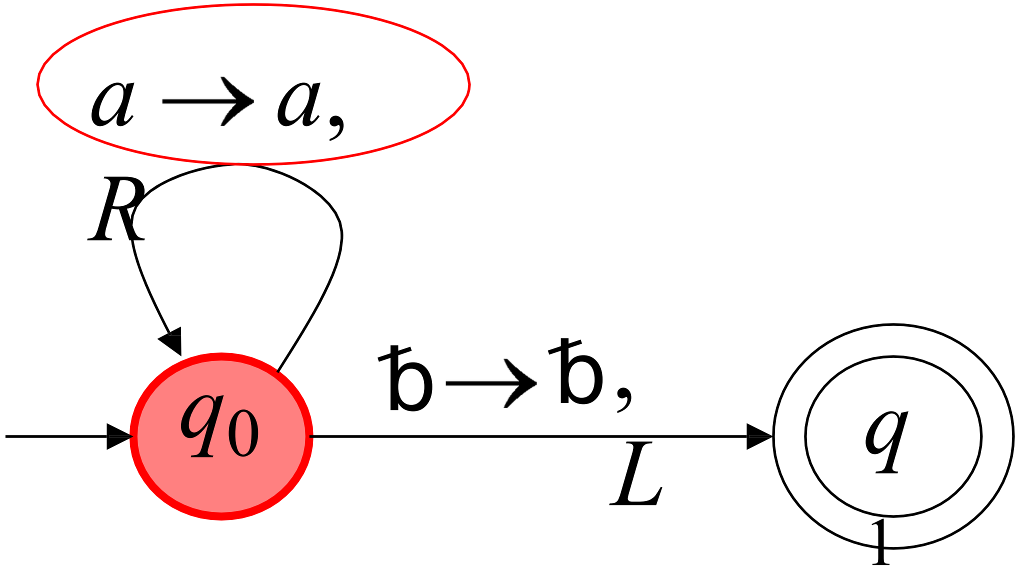



ƀ ƀ a a a ƀ ƀ ƀ
q_0

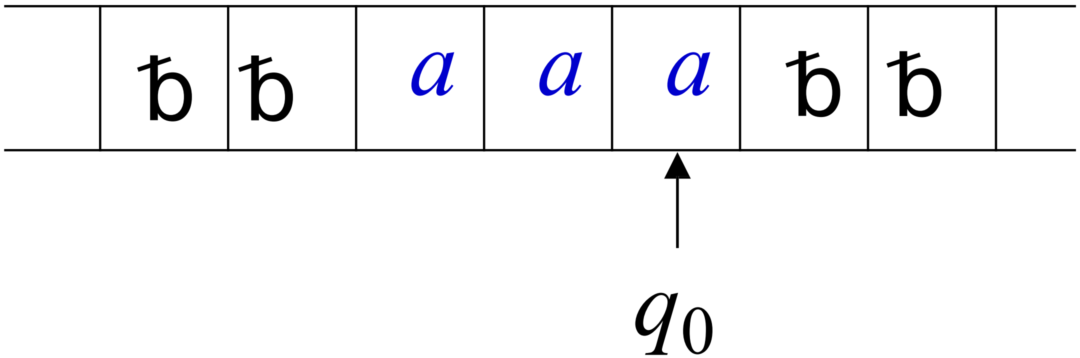

$a \rightarrow a,$
R
ƀ → ƀ,
L
q_0
q_1

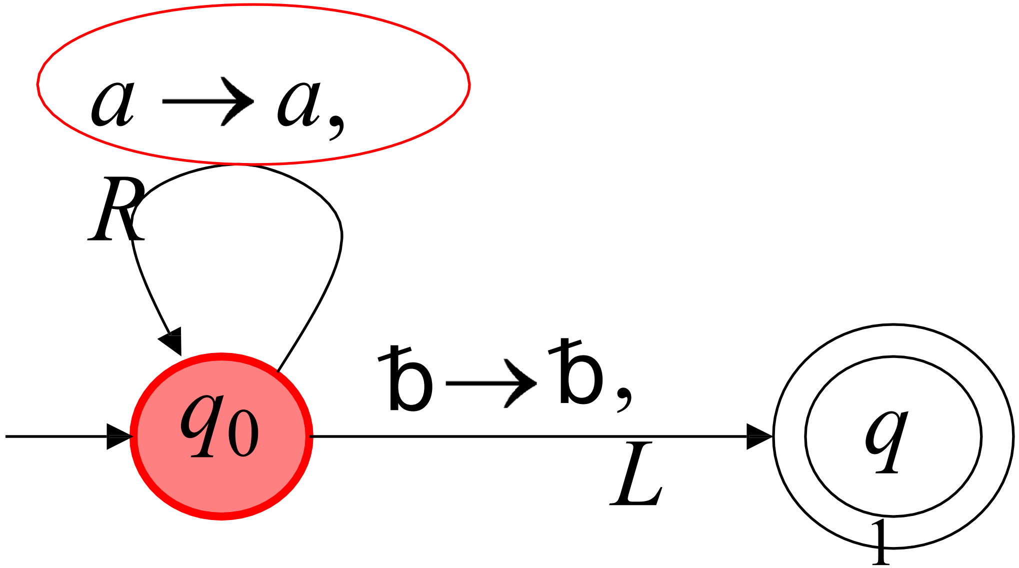


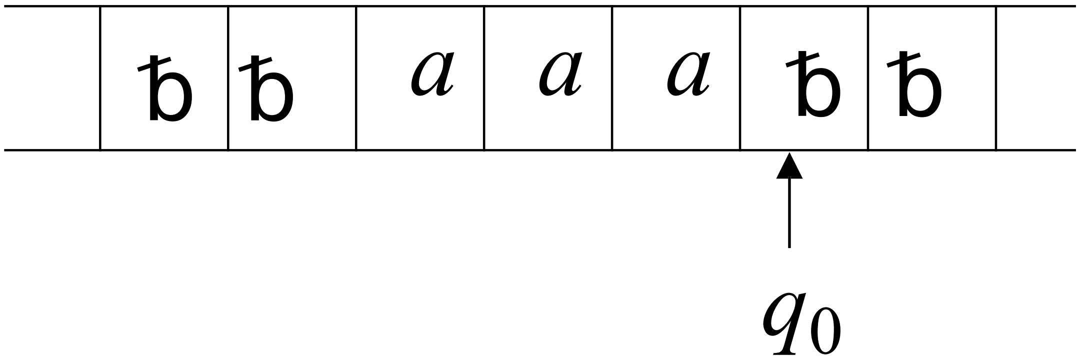
ƀ ƀ a a a ƀ ƀ
q_0

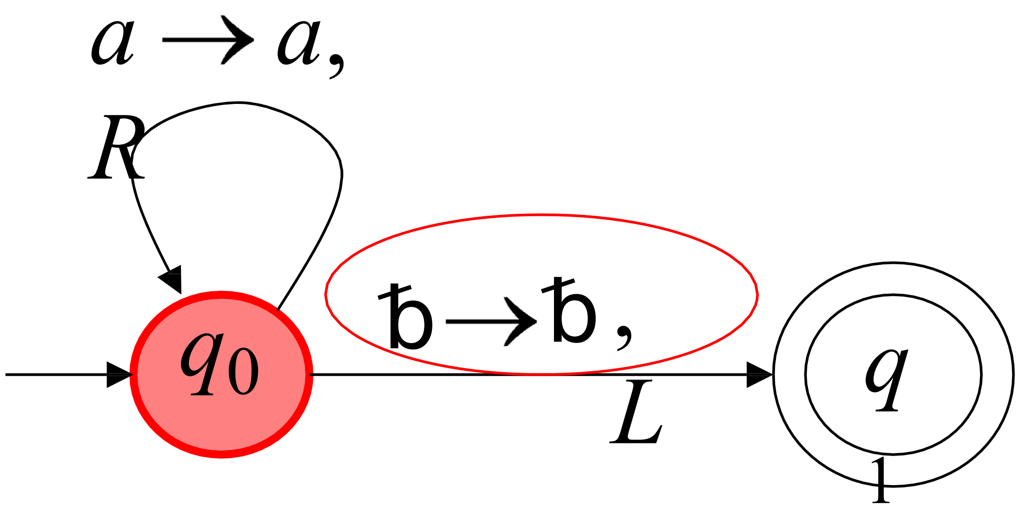


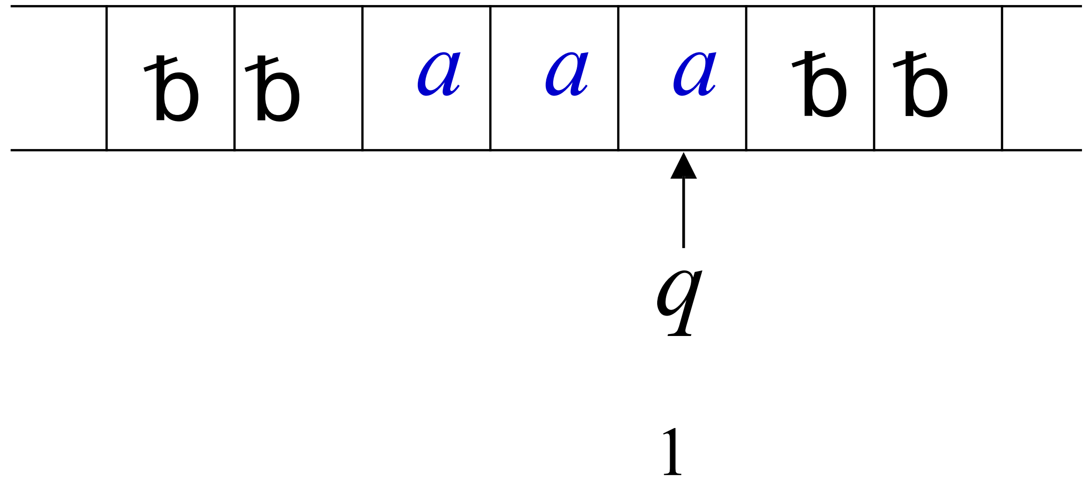
$a \rightarrow a,$
R
q_0
ƀ → ƀ,
L
q_1


ƀ ƀ a a a ƀ ƀ
q_0

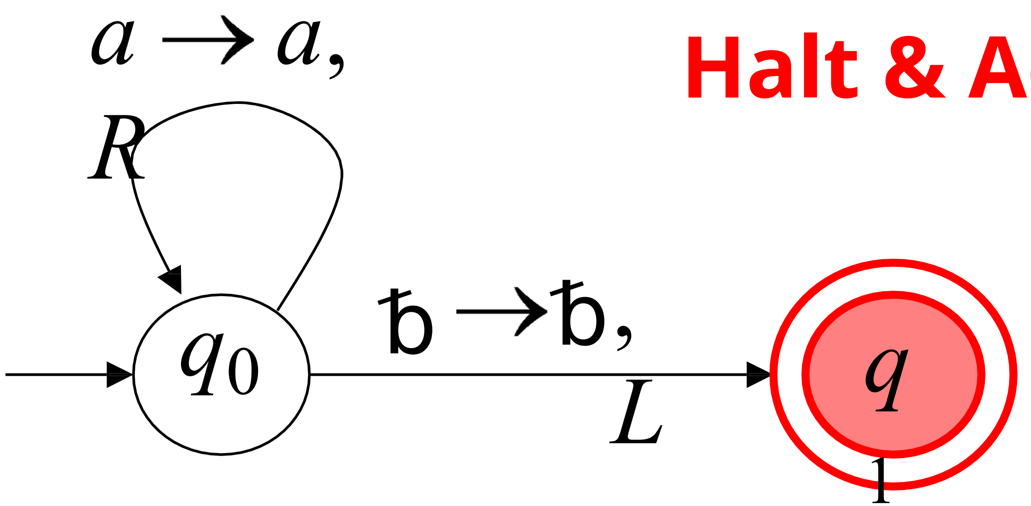

$a \rightarrow a,$
R
ƀ → ƀ,
L
q_0
q_1


ƀ ƀ a a a ƀ ƀ
q0


a → a,
R
ƀ → ƀ,
L
q0
q1

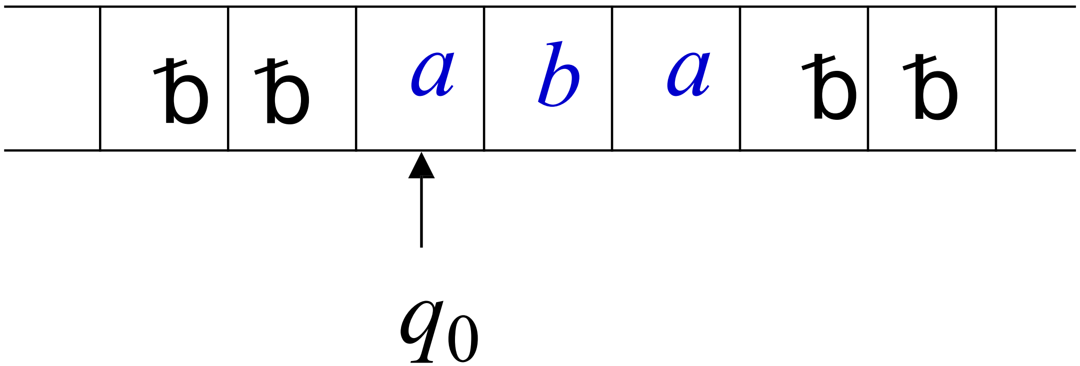


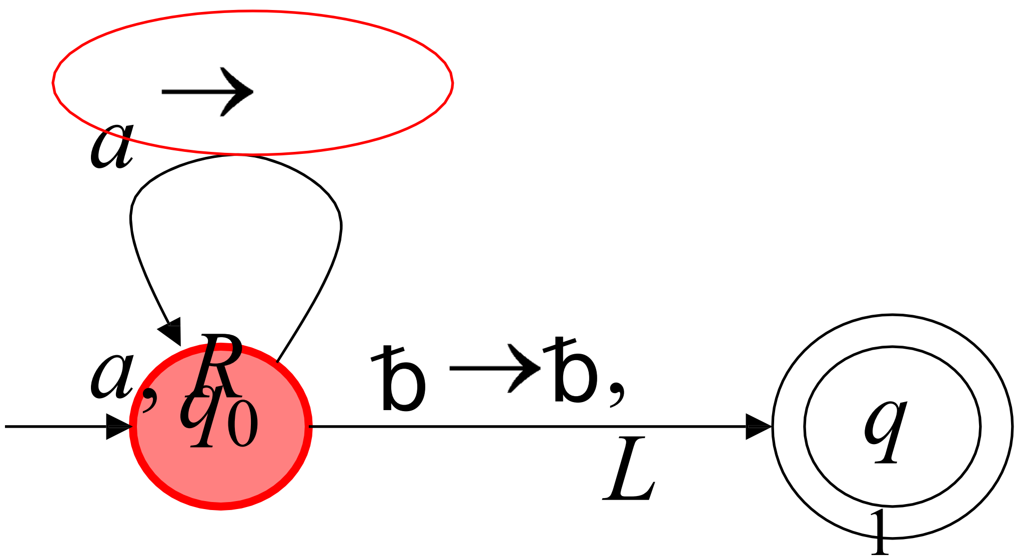
Halt & Accept



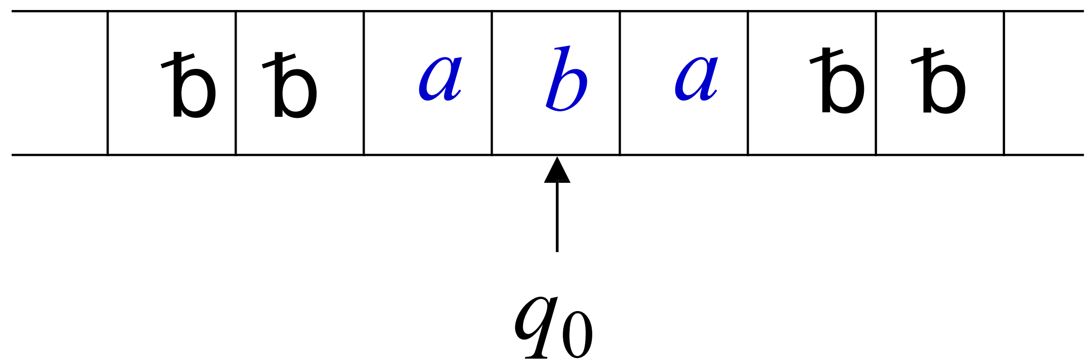
Rejection Example

Time 0



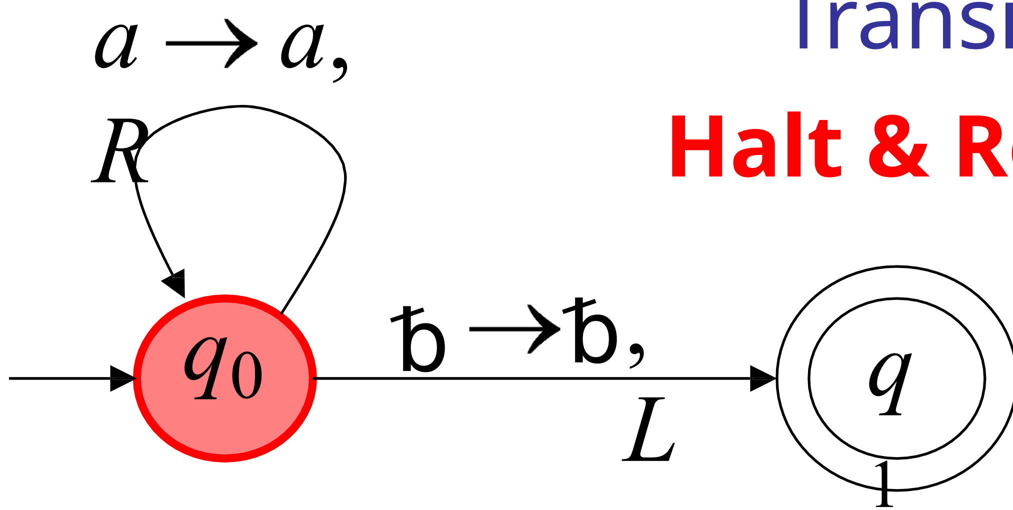


Time 1

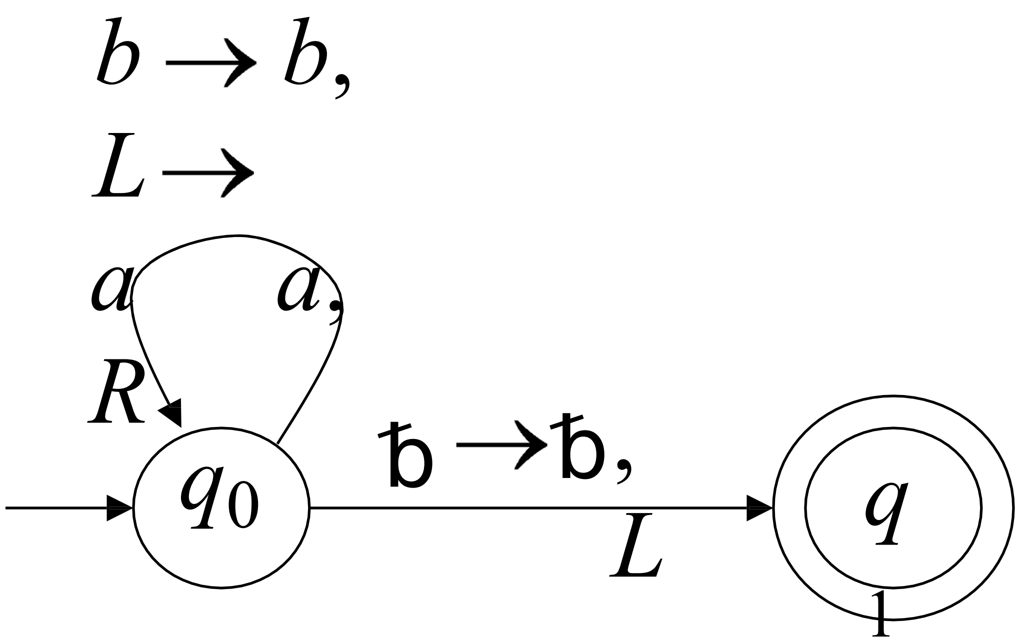


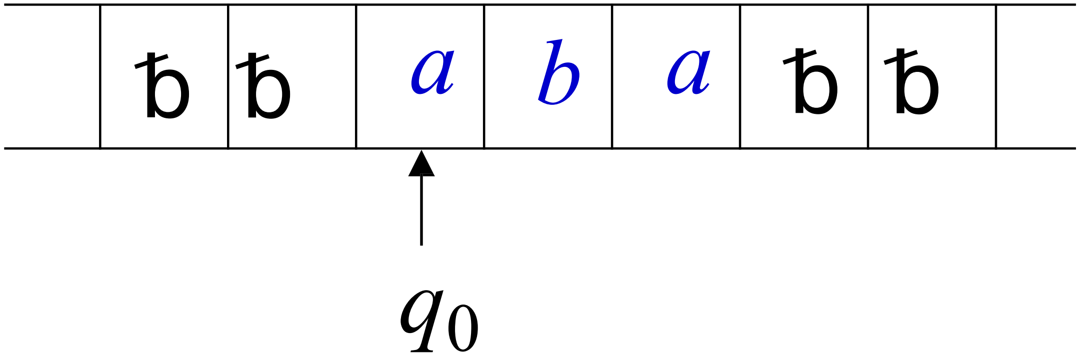
No possible Transition

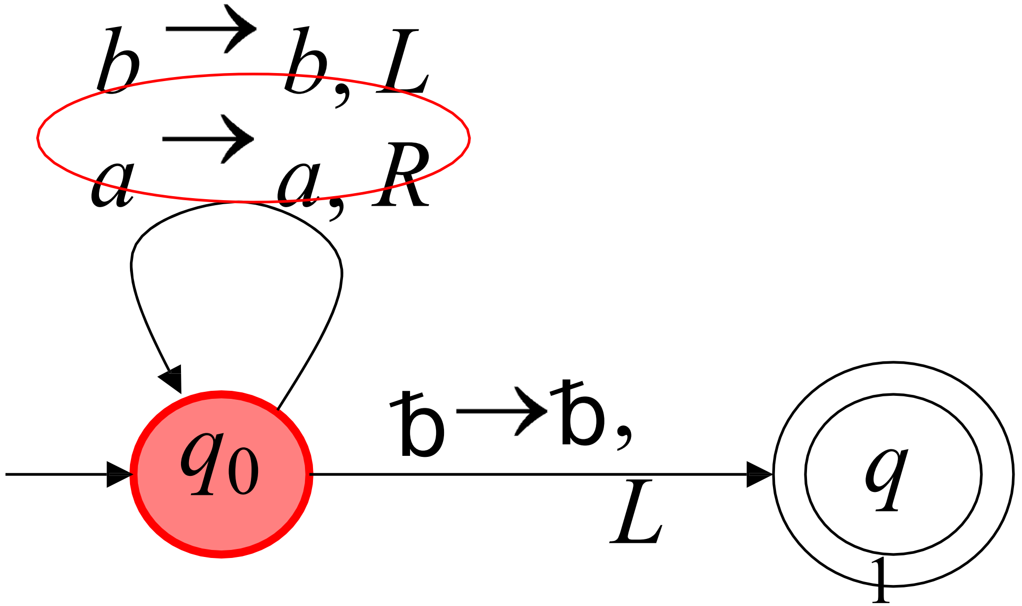
Halt & Reject

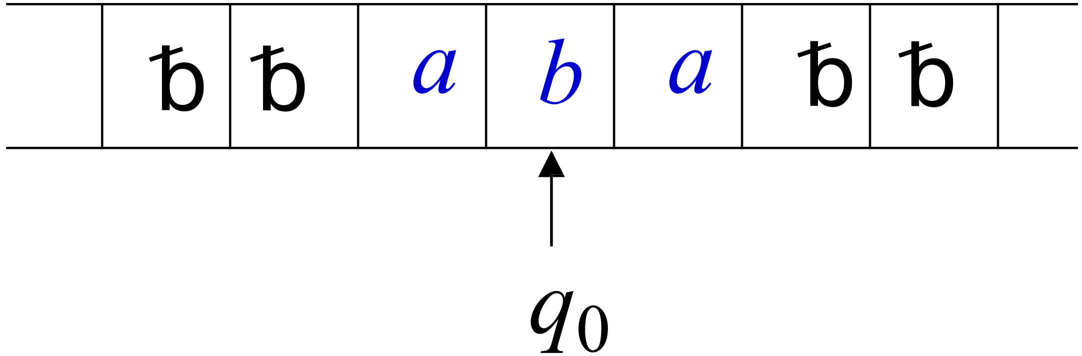


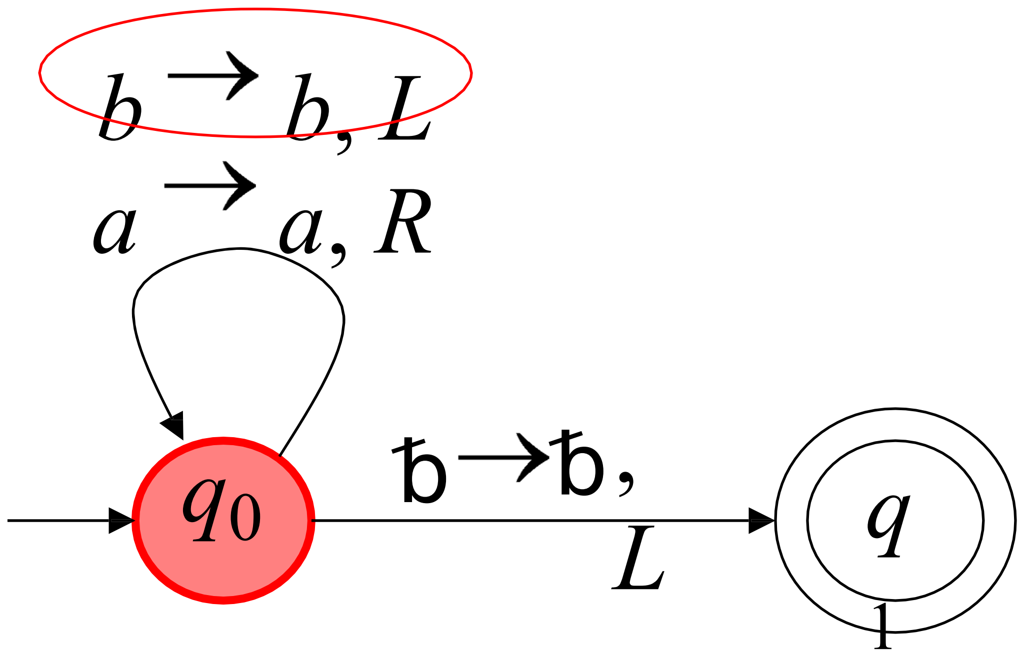
Infinite Loop Example

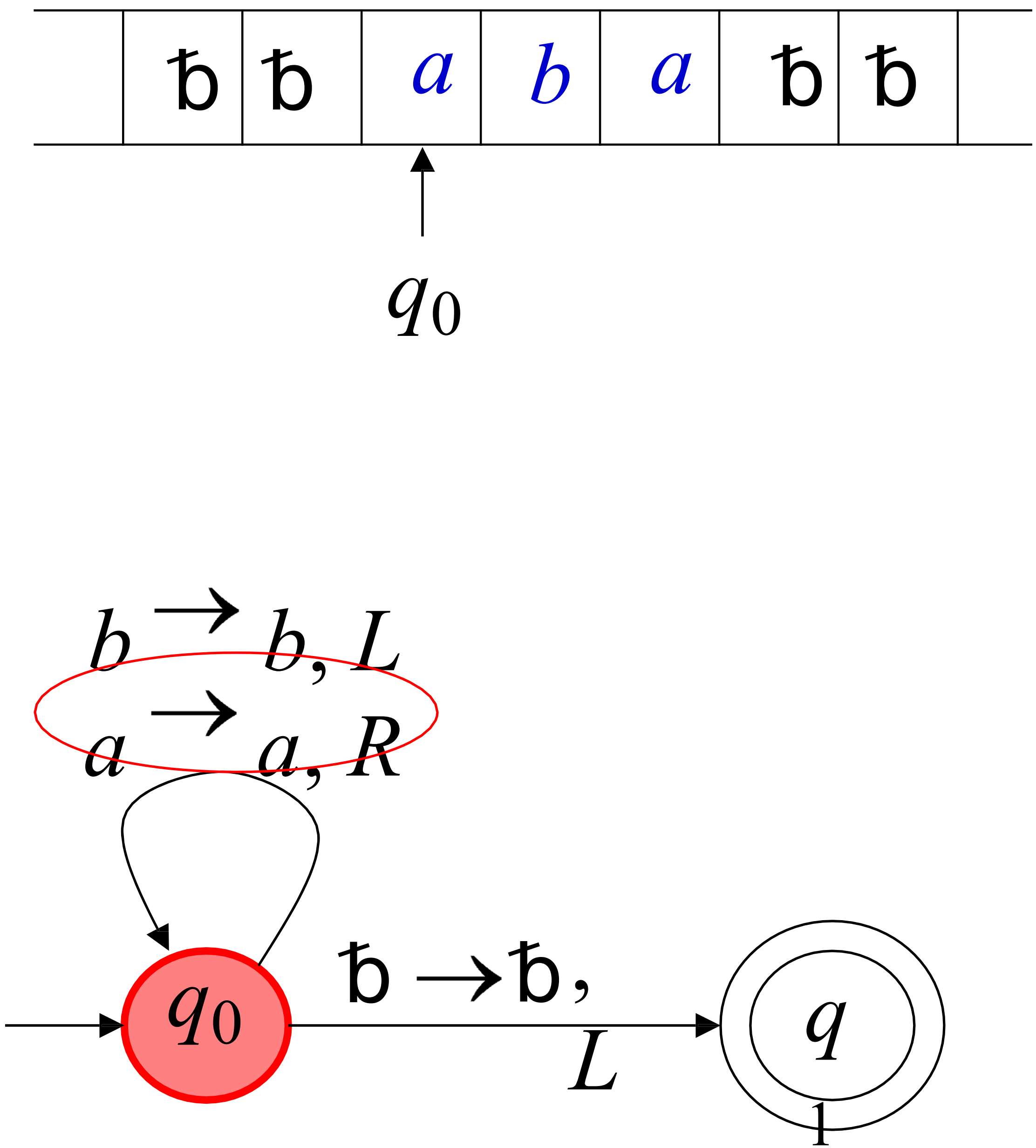



ƀ ƀ a b a ƀ ƀ
q_0

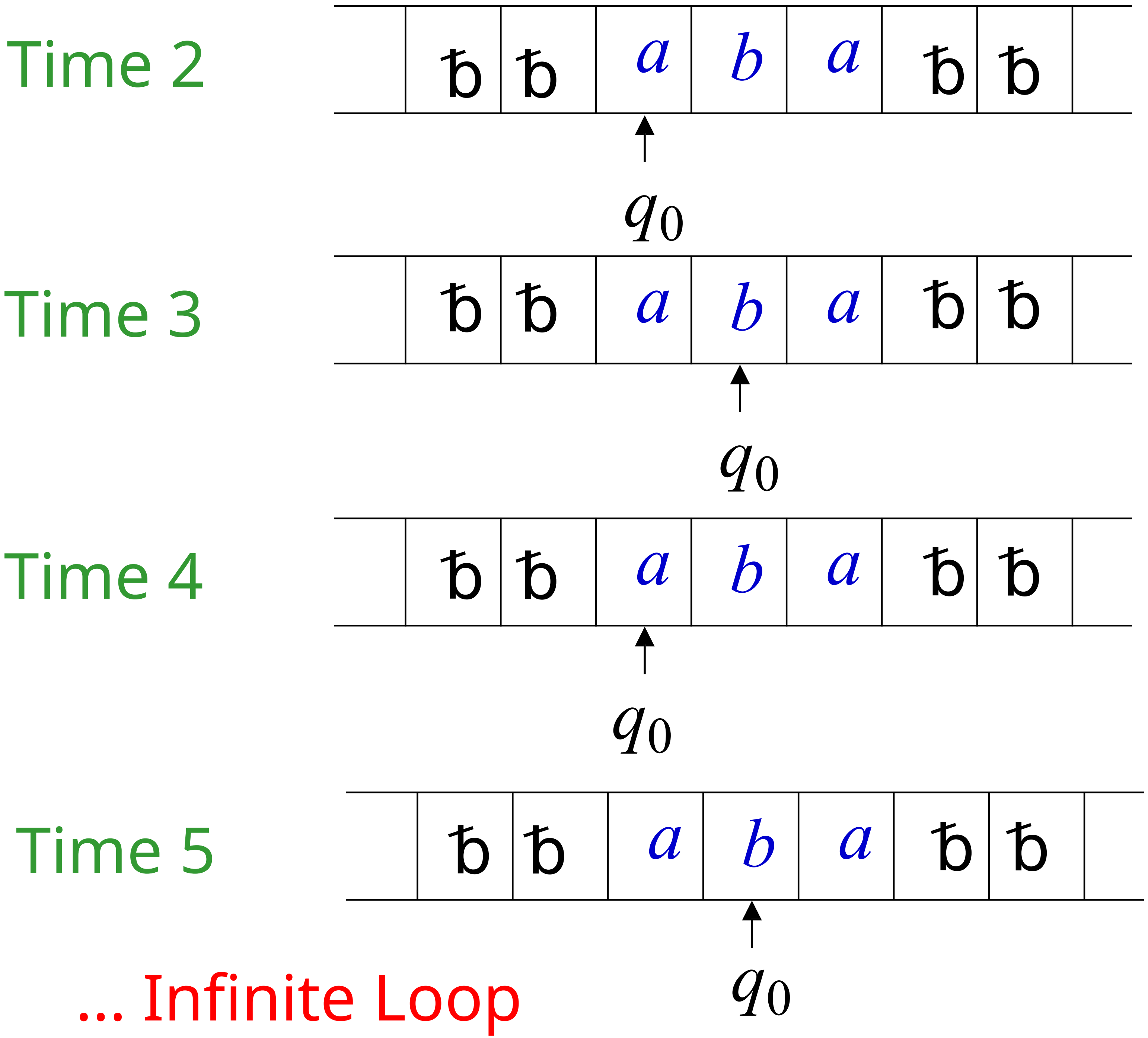

$b \rightarrow b, L$
$a \rightarrow a, R$
q_0
$ƀ \rightarrow ƀ, L$
q_1


ƀ ƀ a b a ƀ ƀ
q0


b → b, L
a → a, R
ƀ → ƀ,
L
q0
q1



ƀ ƀ a b a ƀ ƀ
q_0
$b \rightarrow b, L$
$a \rightarrow a, R$
q_0
ƀ → ƀ, L
q_1


Time 2
ƀ ƀ a b a ƀ ƀ
q_0
Time 3
ƀ ƀ a b a ƀ ƀ
q_0
Time 4
ƀ ƀ a b a ƀ ƀ
q_0
Time 5
ƀ ƀ a b a ƀ ƀ
q_0
... Infinite Loop

Because of the infinite loop:

- The final state cannot be reached
- The machine never halts
- The input is not accepted

Design of TM steps

1. Definition of TM
2. Logic
3. Transition function
4. Instantaneous Description of a string

Turing Machine Examples

1. Design TM to recognize all strings of even number of 1's. Assume the string is made up of only 1's

Solution

$M = (\{q_0, q_1\}, \{1\}, \{1\}, \delta, q0, B, \{q_0\})$

	1	B
$\rightarrow q_0$	q_1BR	Accept
q_1	q_0BR	Reject

Instantaneous Description: w= 1111

$q_0 1111B \vdash B\, q_1 111B \vdash BBq_0 11B \vdash BBB\, q_1 1B \vdash BBBBq_0 B \vdash$ Accept

Turing Machine Examples

2. Design a TM which can compute a concatenation function over $\sum=\{1\}$

If the pair of words {w1,w2} is the input, output has to be w1w2.

B	1	1	B	1	1	1	B

B	1	1	1	1	1	B	B

Solution

$M = (\{q_0, q_1\}, \{1\}, \{1\}, \delta, q0, B, \{q_0\})$

Logic: Replace separating symbol ‘B’ by ‘1’

Replace rightmost ‘1’ by ‘B’

δ:

	1	B
$\rightarrow q_0$	$q_0$1R	$q_1$1R
q_1	q_1 1R	q_2BL
q2	q_3BR	
*q3		Accept

Turing Machine Examples

3. Design TM that will replace every occurrence of substring 11 by 10 keeping everything intact

Solution

$M = (\{q_0, q_1\}, \{0,1\}, \{0,1\}, \delta, q_0, B, \{q_0\})$

	0	1	B
→ q_0	$q_0 0R$	$q_1 1R$	Accept
q_1	$q_0 0R$	$q_0 0R$	Accept

Instantaneous Description: w= 01101110110

4. Design a TM for L= $\{0^n1^n \mid n>=1\}$

	0	1	x	y	B
q_0	q_1, x, R	-	-	q_3, y, R	q_4, B, N(accept)
q_1	q_1, 0, R	q_2, y, L	-	q_1, y, R	-
q_2	q_2, 0, L	-	q_0, x, R	q_2, y, L	-
q_3	-	-	-	q_3, y, R	q_4, B, R(accept)
q_4	-	-	-	-	-

Instantaneous Description: w= 0011

Practice Problems

5. Design TM that recognizes strings containing equal number of 0's and 1's

6. Design TM that checking if a set of parentheses are well-formed

Turing Machine Examples

5. Design TM that recognizes strings containing equal number of 0's and 1's

Solution **Instantaneous Description: w= 01101110110**

$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0,1\}, \{0,1, *\}, \delta, q_0, B, \{q_0\})$

	0	1	*	;
→ q_0	q_1*R	q_3*R	q_0*R	q_4N(Accept)
q_1	$q_1$0R	q_2*L	q_1*R	q_4N(Reject)
q_2	$q_2$0L	$q_2$1L	q_2*L	q_0 ; R
q_3	q_2*L	$q_3$1R	q_3*R	q_4N(Reject)
q_4	-	-	-	-

Turing Machine Examples

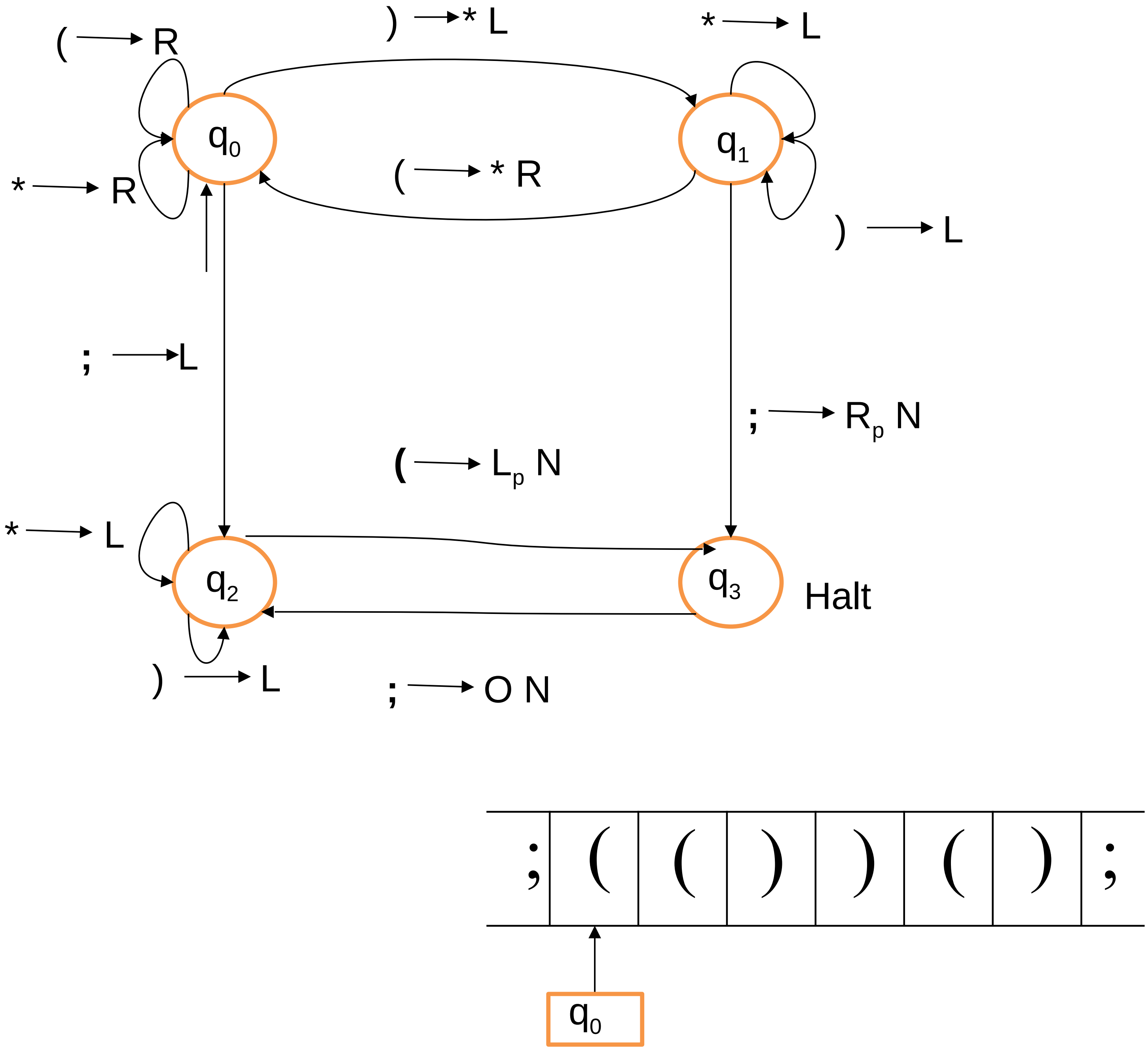
6. Design TM that checking if a set of parentheses are well-formed

$M = (\{q_0, q_1, q_2, q_3,\}, \{(,)\}, \{(,), *, ;, O, R_p, L_p\}, \delta, q_0, B, \{q_3\})$

Simplified Functional Matrix:

	(	*	)	;	Rp	Lp	O
→ q_0	R	R	q_1*L	q_2L	-	-	-
q_1	q_0*R	L	L	q_3R_pN	-	-	-
q_2	q_3L_pN	L	L	q_3ON	-	-	-
q_3	Final state						

Instantaneous Description: w= (( ))


(→ R
) → * L
* → L
q_0
q_1
* → R
(→ * R
) → L
; → L
; → R_p N
(→ L_p N
* → L
q_2
q_3
Halt
) → L
; → O N
; (( )) ( ) ;
q_0

Complexity of Turing Machine

- The complexity of a TM is directly proportional to the size of the functional matrix. In other words, we can say that the complexity of a TM depends on the number of symbols that are being used and the number of states of the TM. Hence:
- Complexity of a TM = $|\Gamma| \times |Q|$ (or $|I| \times |S|$),

  where,

  $|\Gamma|$ = Cardinality of tape alphabet (i.e., number of tape symbols), and

  $|Q|$ = Number of states of the TM.
- For example,

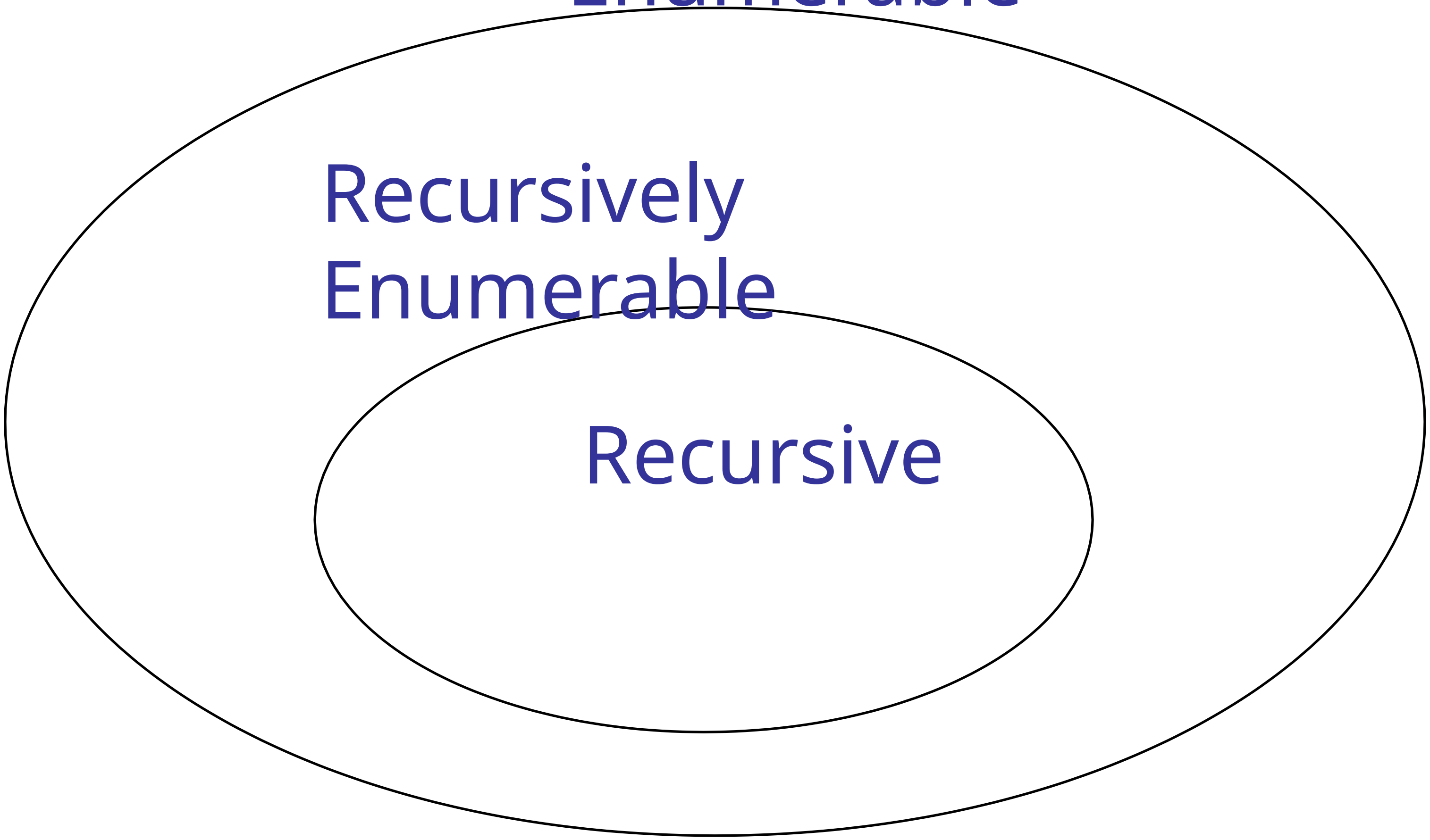
  If $\Gamma = \{1, 0, a, c, ;, ,\}$ and $Q = \{q_0, q_1, q_2, q_3, q_4 = \text{halt}\}$

  Then, the complexity of the TM = $|\Gamma| \times |Q| = 6 \times 5 = 30$

Recursively Enumerable and Recursive Languages

Recursively Enumerable and Recursive Languages

Non Recursively Enumerable



Recursively Enumerable and Recursive Languages

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.

Definition:

A language is **recursive** if some Turing machine accepts it and halts on any input string

In other words:

A language is recursive if there is a membership algorithm for it

Recursive and Recursively Enumerable Languages

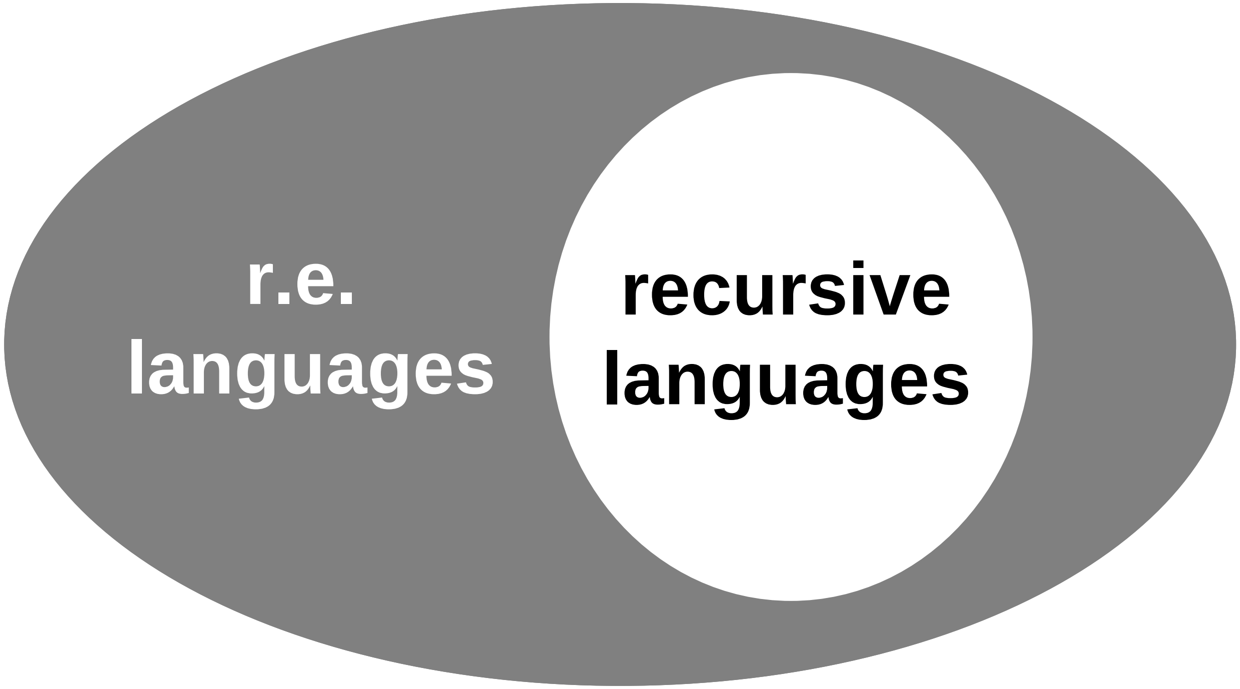
To summarize we can say that,

- **Recursively Enumerable Set**
  - A set S of words over Σ is said to be recursively enumerable, if there is a TM over Σ, which accepts every word in S and either rejects or loops for every word in $\sim S$ ($\sim S = \Sigma^* - S$). This can be represented as:
    - Accept TM = S
    - Reject (TM) $\cup$ loop (TM) = $\Sigma^* - S$
- **Recursive Set**
  - A set S of words over Σ is said to be recursive, if there is a TM over Σ, which accepts every word in S and rejects every word in $\sim S$ ($\sim S$ $= \Sigma^* - S$). This can be represented as:
    - Accept (TM) = S
    - Reject (TM) = $\Sigma^* - S$
    - Loop (TM) = ϕ

Recursively Enumerable and Recursive Languages

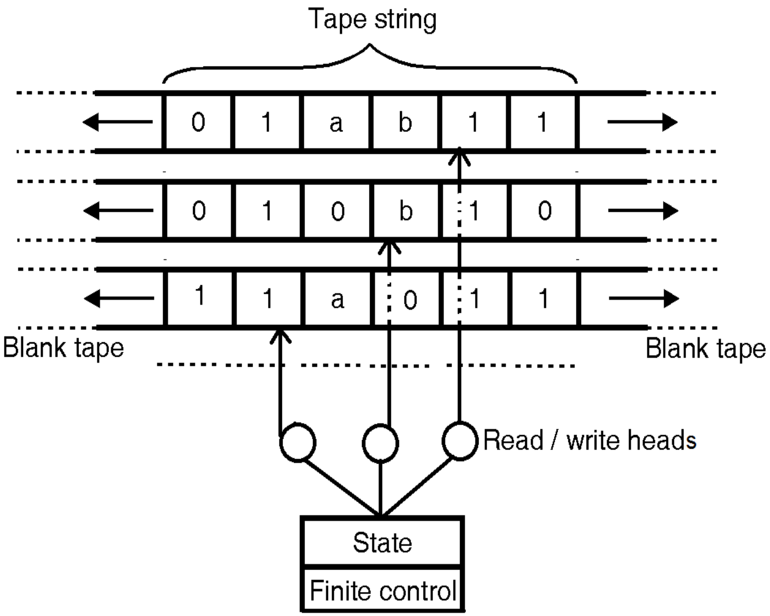
A language is called Turing-recognizable or recursively enumerable (r.e.) if some TM recognizes it

A language is called decidable or recursive if some TM decides it



Multi Tape Turing Machine

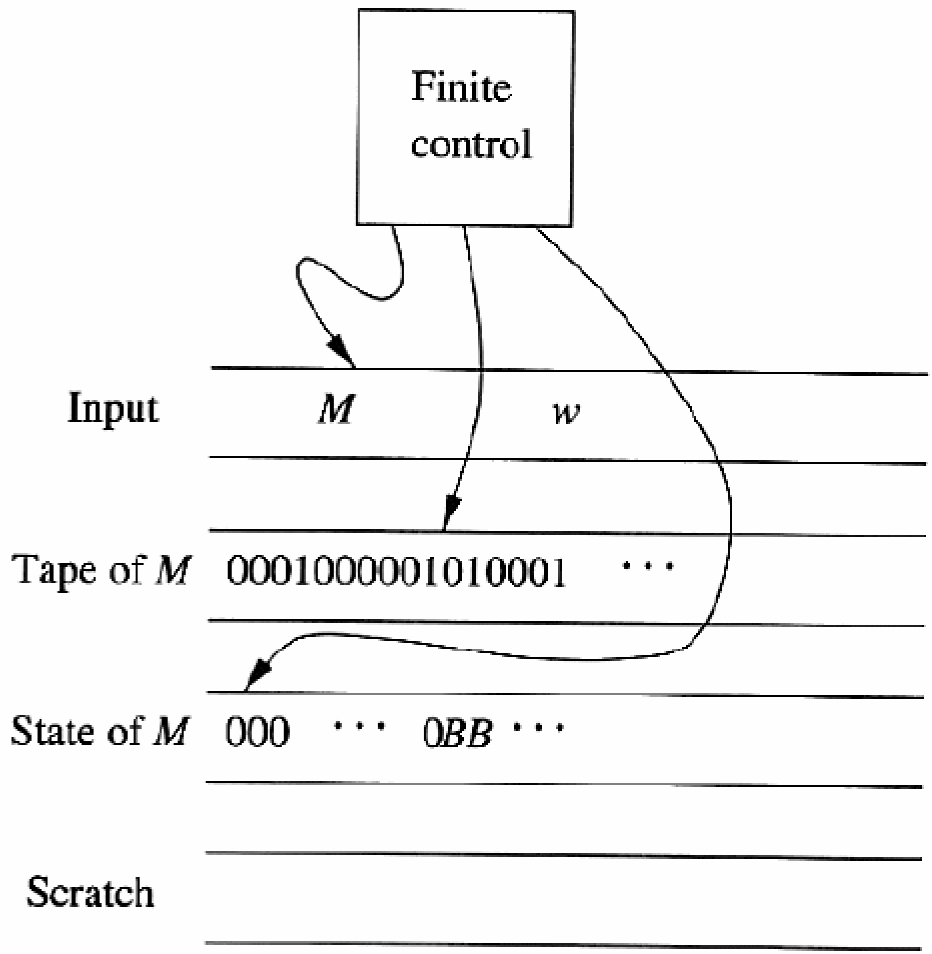
- Multi-tape Turing machines have *k* number of independent tapes, having their own read/write heads. These machines have independent control over all the heads—any of these can move and read/write their own tapes. All these tapes are unbounded at both the ends just as in the single- tape TM.
- Multi-tape TM and single-tape TM are equivalent in power (except for some difference in execution time



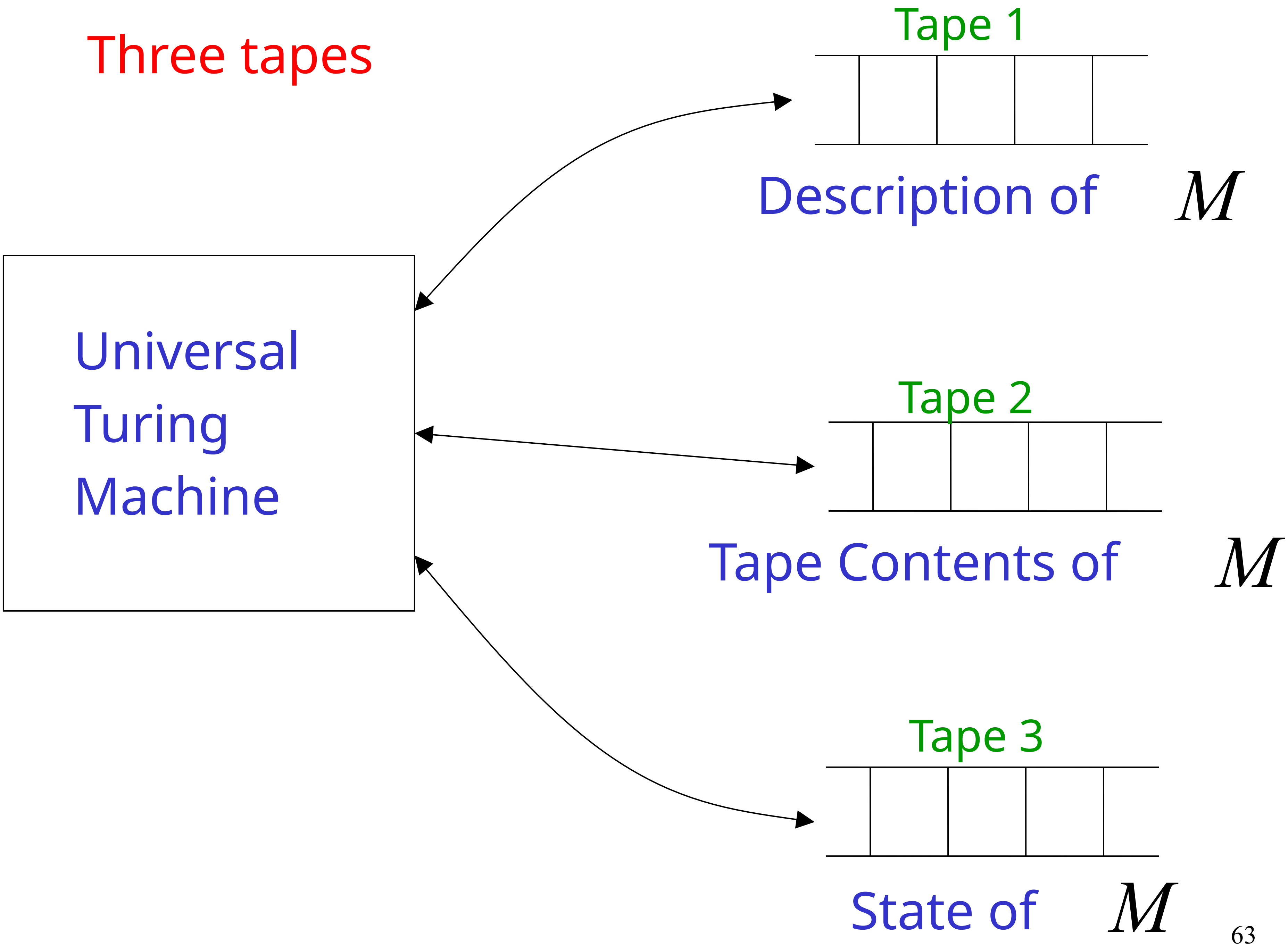
Universal Turing Machine

- **Universal Turing Machine** (**UTM**)
  - Turing machine that can simulate any other Turing Machine
  - It accepts the *encoded* Functional Matrix of any other TM as input on its tape (**Program area**)
  - It also accepts the data on which the other TM needs to be simulated (**Data area** of the tape)
  - UTM needs an **imitation algorithm** that can simulate the functional matrix of any other TM (**System area**)
  - Functional Matrix of such a UTM is analogous to an Operating System
  - **UTM is analogous to a modern Computer!**

Universal Turing Machine



Three tapes


Tape 1
Description of M
Universal
Turing
Machine
Tape 2
Tape Contents of M
Tape 3
State of M

Non-Deterministic Turing Machine

A nondeterministic Turing Machine (NTM) differs from the deterministic variety by having a transition function δ such that for each state q and tape symbol X, $\delta(q, X)$ is a set of triples

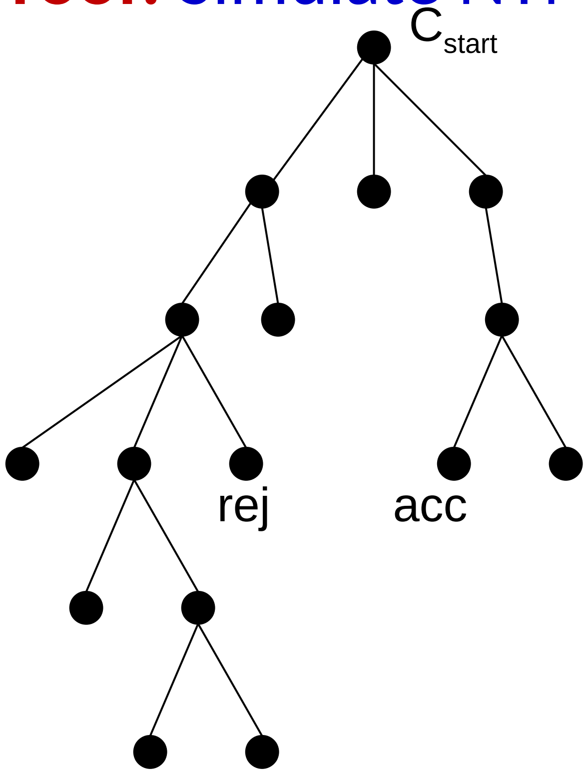
$$\{(q_1,Y_1,D_1), (q_2,Y_2,D_2), \ldots\ldots\ldots\ldots (q_k,Y_k,D_k)\}$$

Where k is any finite integer. The NTM can choose, at each step, any of the triples to be the next move. It cannot, however, pick a state from one, a tape symbol from another, a the direction from yet another.

NTM and DTM

Theorem: Every NTM has an equivalent (deterministic) TM

Proof: Simulate NTM with a deterministic TM



- computations of M are a tree
- nodes are configurations
- fanout is b = maximum number of choices in transition function
- leaves are accept/reject configurations.

Simulating NTM M with a deterministic TM:

Breadth-first search of tree

- if M accepts: we will encounter accepting leaf and accept
- if M rejects: we will encounter all rejecting leaves, finish traversal of tree, and reject
- if M does not halt on some branch: we will not halt as that branch is infinite…

Simulating NTM M with a deterministic TM:

- use a 3 tape TM:
  - tape 1: input tape (read-only)
  - tape 2: simulation tape (copy of M's tape at point corresponding to some node in the tree)
  - tape 3: which node of the tree we are exploring (string in $\{1,2,\ldots b\}^*$)
- Initially, tape 1 has input, others blank

NTM and DTM

Here is the transition function of a nondeterministic TM $M = (\{q_0, q_1, q_2\}, \{0,1\}, \{0,1,B\}, \delta, q_0, B, \{q_2\})$:

δ	0	1	B
q_0	$\{(q_0, 1, R)\}$	$\{(q_1, 0, R)\}$	$\emptyset$
q_1	$\{(q_1, 0, R),\ (q_0, 0, L)\}$	$\{(q_1, 1, R),\ (q_0, 1, L)\}$	$\{(q_2, B, R)\}$
q_2	$\emptyset$	$\emptyset$	$\emptyset$

Show the ID's reachable from the initial ID if the input is:

* a) 01.

b) 011.

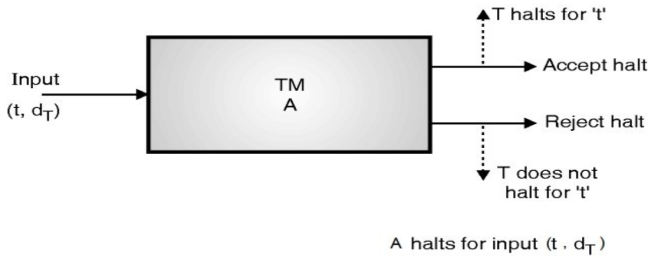
Solvable and Semi-Solvable Problems

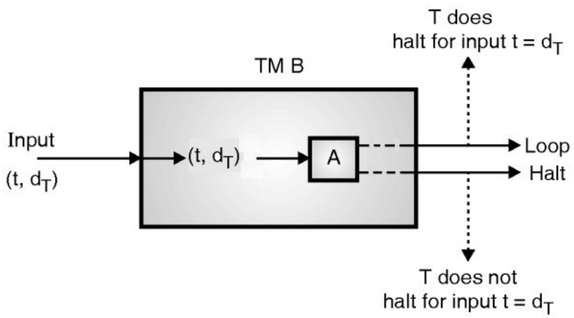
- **Solvable** problem: TM when applied to such a problem, always eventually terminates with the correct “yes” or “no” answer
  - A class of all such problems is called as **Recursive language**
  - The mathematical functions that denote these type of problems are called as **Total Recursive Functions**
  - Simple Examples - multiplication, addition, concatenation and many other

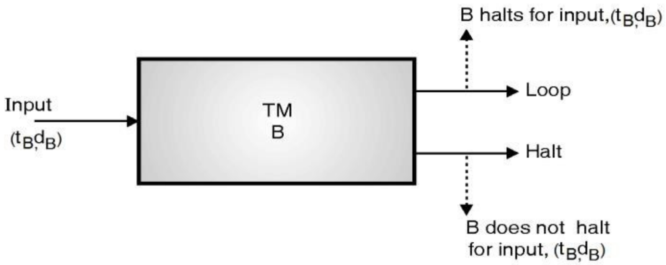
- **Semi-solvable** problem: TM when applied to such a problem, always eventually terminates with correct answer when answer is “yes” and may or may not terminate when the correct answer is “no”
  - A class of all such problems is called as **Recursively Enumerable language**
  - The mathematical functions that denote these type of problems are called as **Partial Recursive Functions**
  - Simple Examples - division, factorial and many other

Halting Problem and Unsolvablity

- For a given input for any general TM two cases arise,
  - The machine may halt after a finite number of steps
  - The machine may not ever halt no matter how long it runs
- Given any TM, problem of algorithmically determining whether it ever halts or not, is called as the **Halting Problem**
- The halting problem is **unsolvable**

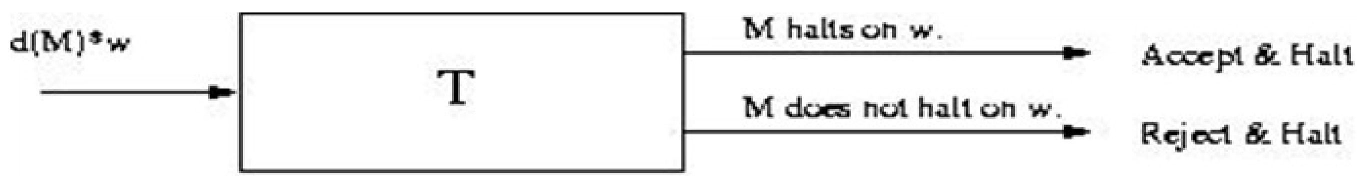






Proof :-Halting problem is undecidable

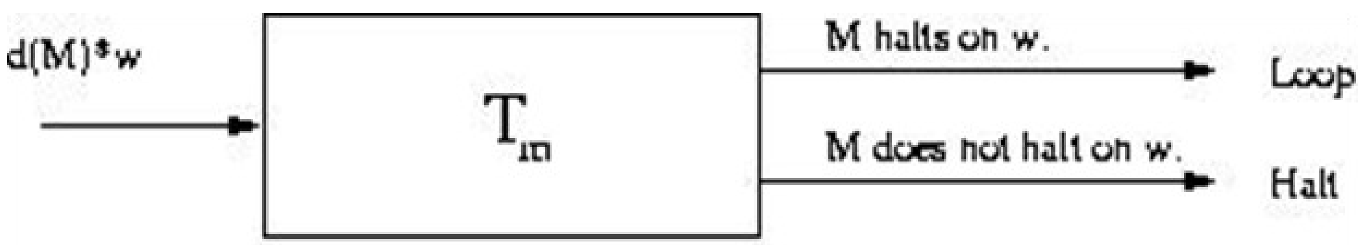
- Proof by contradiction
- There is a Turing machine T that will decide the halting problem.<M> this is the description of Turing machine M and string W . T write "accept" when M halts on w, and reject If M does not halts on W



Turing machine T

Proof :-Halting problem is undecidable

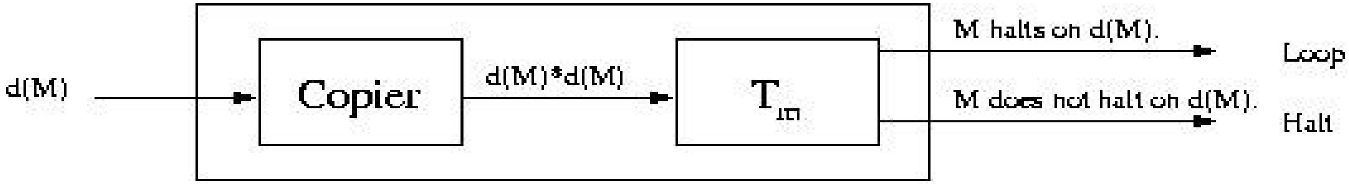
We build a Turing machine Tm and here we standardizing T so when T write yes and halt then Tm will goes into loop forever



Turing machine T_m

Proof :-Halting problem is undecidable

we build Tc with the help of TM. Here we take Tc as input which is the description of Turing machine M and we write it in this way d(M) now we copies it to obtain the string d(M)*d(M), where * is a symbol that break up the two copies of d(M) and then provide d(M)*d(M) to the Turing machine Tm .



Turing machine T_c

Proof :-Halting problem is undecidable

What Turing machine Tc does when a string given to it which describe Tc itself

d(Tc) is given as input to Tc it make copy of it and build the string d(Tc)*d(Tc) and allot to standardized T.so the altered T is specified a description of Tc and string d(Tc)

Turing machine T_c on input d(T_c)

END