TURING MACHINES

(Unrestricted Language)

Alain Turing

1912-1954

- studied problems which today lie at the heart of artificial intelligence.
- proposed the Turing Test which is still today the test people apply in attempting to answer whether a computer can be intelligent.

Computing machinery and intelligence

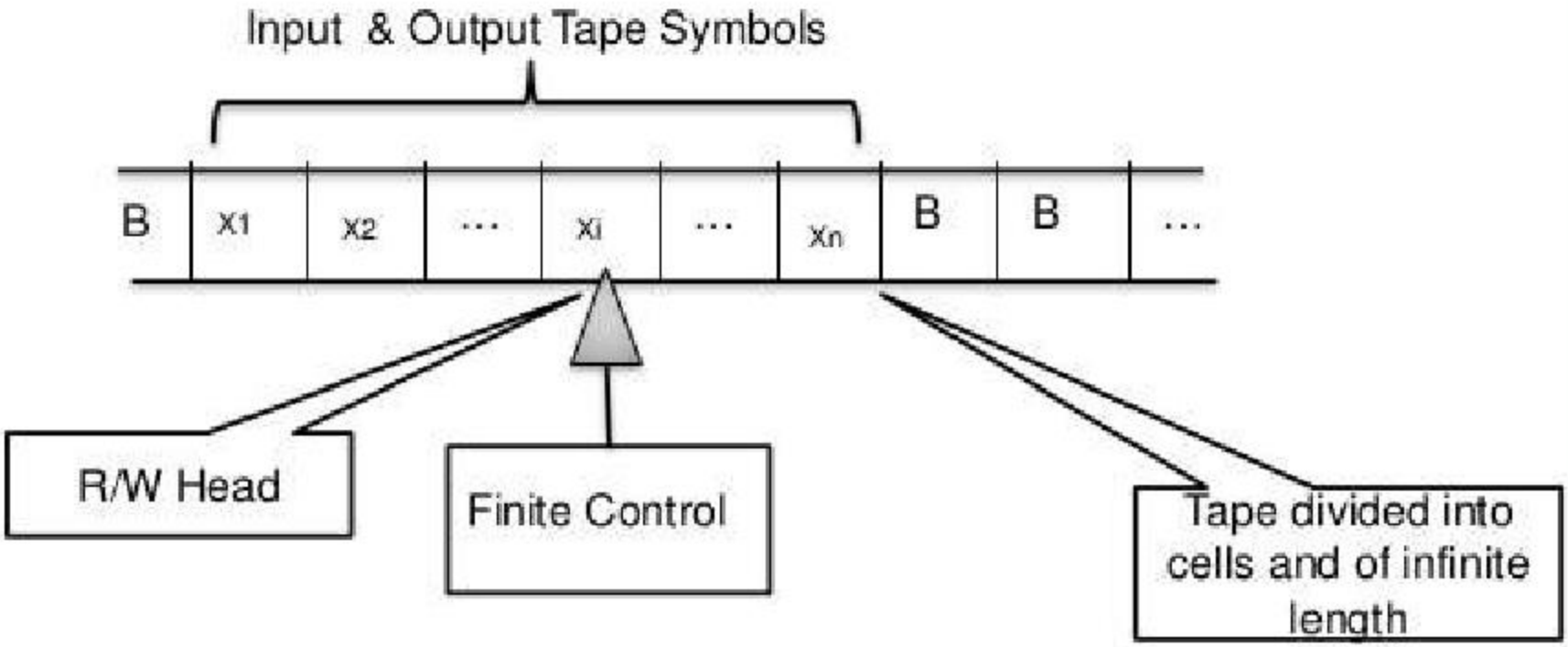
INTRODUCTION

- TM Invented by esteemed computer scientist **ALAN TURING** in 1936
- Basically an abstract computational model
- Provides a powerful computational model for solving problems and functions in computer science
- Capable of simulating common computers

WHAT IS A TURING MACHINE ?

- Consists of an infinite tape (as the memory) Used as Input as well as Output storage device.
- A tape head (a pointer to the currently inspected cell of the memory)
- And a state transition table(to govern the behavior of the machine)

THE TURING MACHINE MODEL



B is blank/Null/ϵ/ Δ Symbol

Assumed that tape is infinite in both directions

Description

- A **tape is** divided into cells, one next to the other. Each cell contains a symbol from some finite set.
- A **head** that can read and write symbols on the tape and move on the tape left and right one cell at a time.
- There is **a state control**, which can be in any one of a finite number states.
- The finite set of states is denoted by Q. The set Q contains three special states:

  1) a start state q0

  2) an accept halt state ha → TM halts on accept, ie. halt_accept

  3) a reject state hr → TM halts on reject, ie. halt_reject

- If the tape symbol contains the blank symbol B or Δ, it means that the cell is actually empty.

FORMAL DEFINITION

TM is a five-tuple: $M = (Q, \Sigma, \Gamma, \delta, q_0)$

Q	A finite set of states
Σ	A finite set of input symbols
Γ	A finite set of tape symbols, $\Sigma \subseteq \Gamma$ (B is a distinguished blank symbol, $B \in \Gamma$)
q_0	The initial/starting state, $q_0 \in Q$
δ	A next-move, i.e.*mapping or transition function*

$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$

Head Movement → L= Left one cell; R= Right one cell

δ is a partial function. The value of $\delta(q,X)$ is either undefined, or is a triplet consisting of the new state, the replacement symbol, and direction (left/right) for the head motion

Definition_alternate

A *Turing machine* (TM) is a 5-tuple $T = (Q, \Sigma, \Gamma, q_0, \delta)$, where

Q is a finite set of states, assumed not to contain h_a or h_r, the two *halting* states (the same symbols will be used for the halt states of every TM);

Σ and Γ are finite sets, the *input* and *tape* alphabets, respectively, with $\Sigma \subseteq \Gamma$; Γ is assumed not to contain Δ, the *blank* symbol;

q_0, the initial state, is an element of Q;

$\delta : Q \times (\Gamma \cup \{\Delta\}) \rightarrow (Q \cup \{h_a, h_r\}) \times (\Gamma \cup \{\Delta\}) \times \{R,L,S\}$ is a partial function (that is, possibly undefined at certain points).

TM - Deterministic in behavior

WORKING

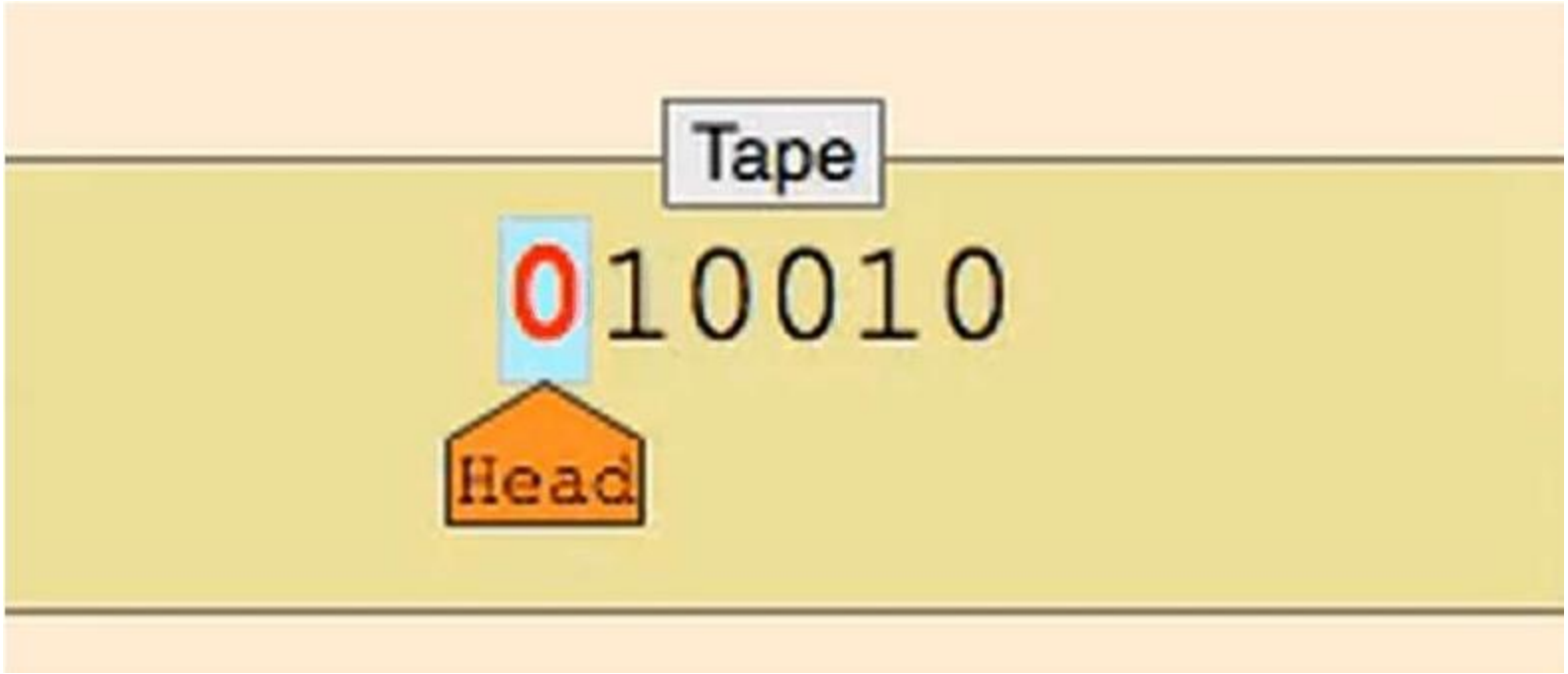
The machine operates on an infinite memory tape divided into discrete cells. The machine positions its head over a cell and reads the symbol there. Then, as per the symbol and its present place in user specified instructions , the machine

1. Writes a symbol in the cell, then
2. Either moves the tape one cell right or left
3. Either proceeds to a subsequent instruction or halts the computation

By Default, TM is **deterministic** in nature. However, TM determinism is a **partial function**. In other words, TM has **at most one move from any state with a specific symbol** as a input. **For some combinations of state and input symbol, Machine have no transition** (Property of partial function)

EXAMPLE : TM THAT CHECKS FOR STRING

PALINDROME Input on the tape with head



Implicit Assumptions

- Input is placed on tape in contiguous block of cells
- All other cells to the right are blank: ‘B’
- Tape head positioned at Left of input block
- There is only one start state
- The text uses a single accepting and possibly many rejecting states. If multiple reject states are available, they are considered equivalent.

A TM can compute an output by storing an answer on the tape when it halts with accept.

PROPERTIES of TM

- **Recognizability :** A language is recognizable if a TM accepts when an input string is in the language, and either rejects or loops forever when an input string is not in the language.

- **Decidability :** A language is decidable if a TM accepts strings that are in the language and rejects that are not in the language. That is, TM will halt (halt accept) on all inputs. It gives confirm answer on any number of iterations.

- **The Halting Problem** is recognizable but undecidable. (There is no unique observation why TM enters in halt reject condition)

The instantaneous Description(ID) of TM can be given by a triplet or a program as:

$$(q0,a) \rightarrow (q1,A,L)$$

That means currently TM is reading the input Symbol ‘a’ and is in q0 state. TM can go to q1 state by replacing or printing ‘a’ by A and moving in left direction.

TM- Move & ID notation

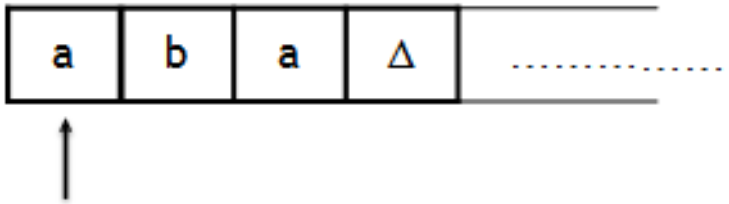
$$\delta(q, a) = (r, \Delta, \mathrm{L})$$

$$(q, aab\underline{a}\Delta a) \vdash_T (r, aa\underline{b}\Delta\Delta a)$$

- Example:

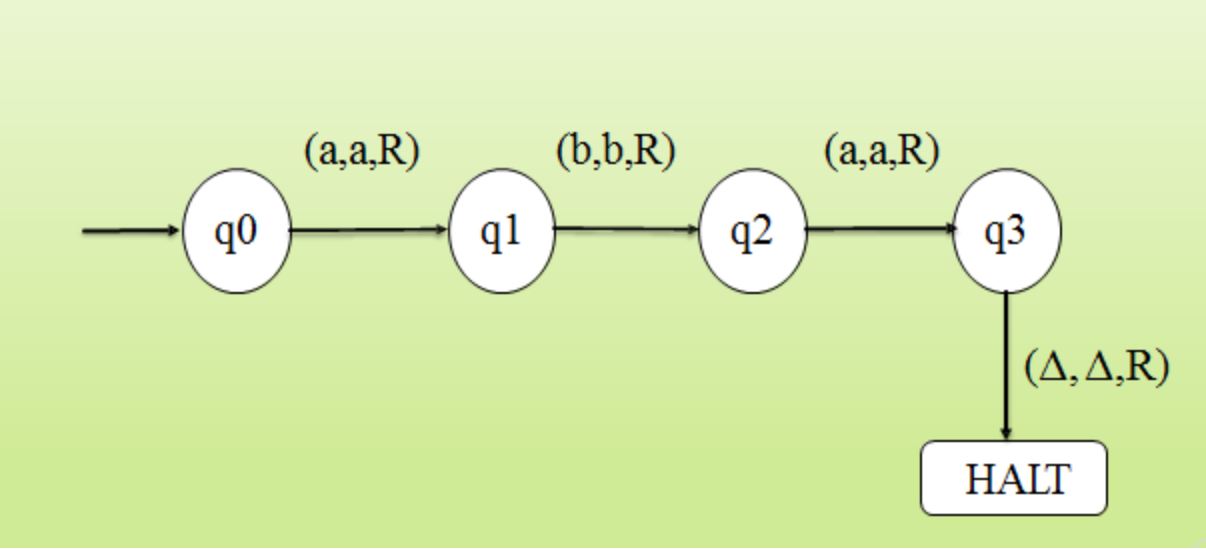
Construct a TM that accepts the language of ‘aba’ over ∑={a,b}.

Assume that, on the input tape the string ‘aba’ is placed like:

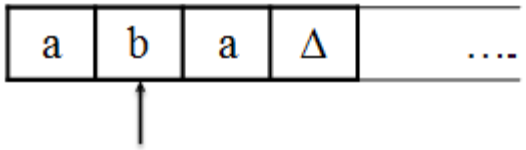


The tape head will read out the sequence up to the character Δ if ‘aba’ is readout TM will halt after reading Δ. (Δ is blank)

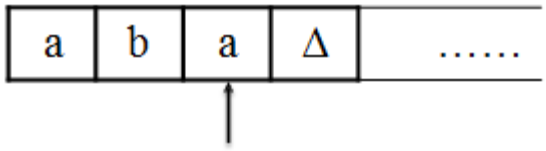
Example Continued:



(Triplet along the edge:(input read ,output to be printed , direction of head move)



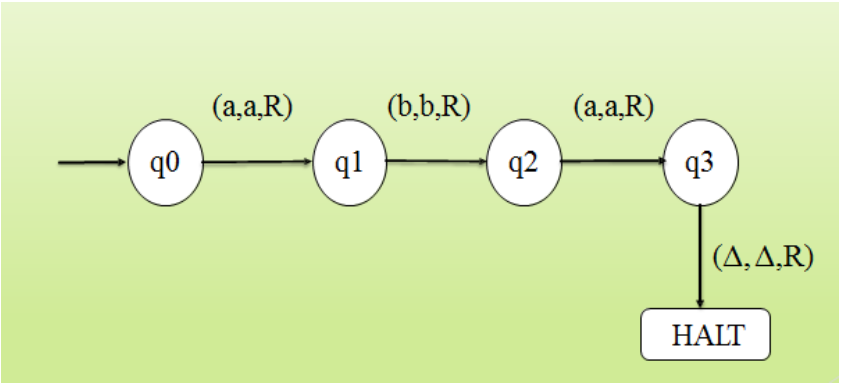
Consider the transition between start state and q1 which is (a,a,R) that is the current symbol read from the tape is a then output it as a, and move the tape head to the right.



And then after transition between q1 to q2..

The TM will accept the language when it reaches to halt_accept state

Example Continued:



	a	b	Δ
start	(q1,a,R)	-	-
q1	-	(q2,b,R)	-
q2	(q3,a,R)	-	-
q3	-	-	(HALT, Δ,R)
HALT	-	-	-

In the given transition table,

a triplet is written in each raw as:

(next state , output to be printed , direction)

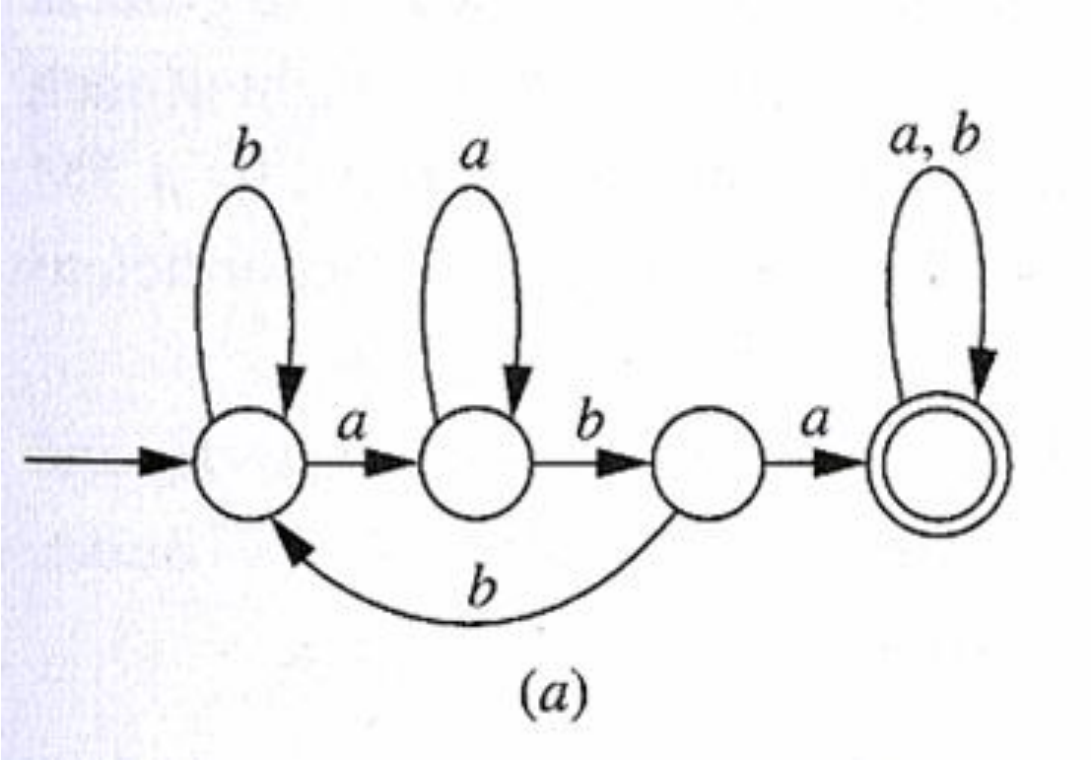
TM - Acceptance

If $T = (Q, \Sigma, \Gamma, q_0, \delta)$ is a Turing machine, and $x \in \Sigma^*$, x is accepted by T if, starting in the initial configuration corresponding to input x, T eventually reaches an accepting configuration. In other words, x is accepted if there exist $y, z \in (\Gamma \cup \{\Delta\})^*$ and $a \in \Gamma \cup \{\Delta\}$ so that

$$(q_0, \underline{\Delta}x) \vdash_T^* (h_a, y\underline{a}z)$$

The *language accepted by* T is the set $L(T)$ of input strings accepted by T.

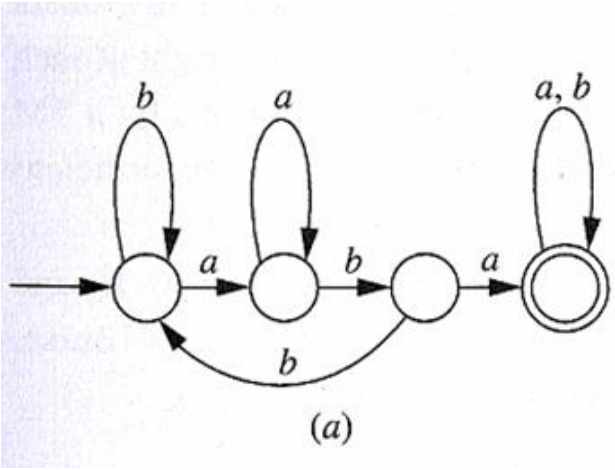
(a)DFA to accept $\{a,b\}^*\{aba\}\{a,b\}^*$

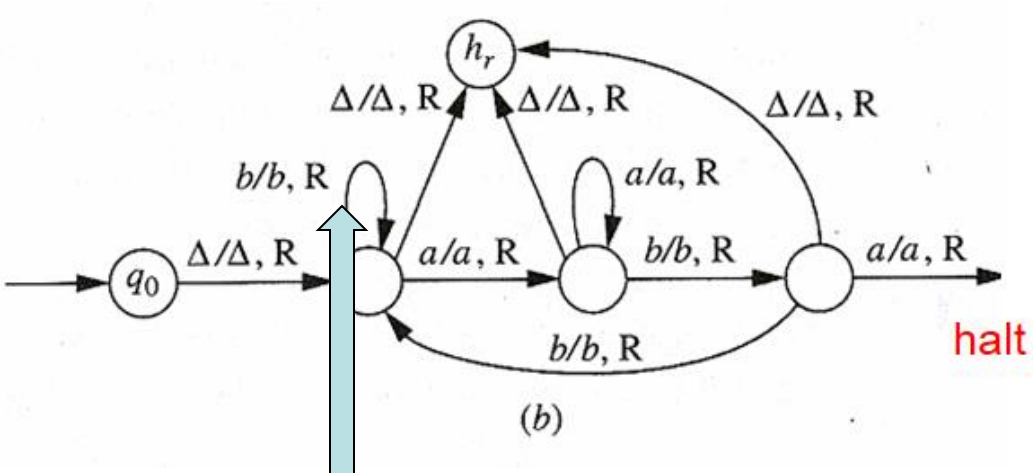


(a)DFA

(b)TM to accept $\{a,b\}^*\{aba\}\{a,b\}^*$

?

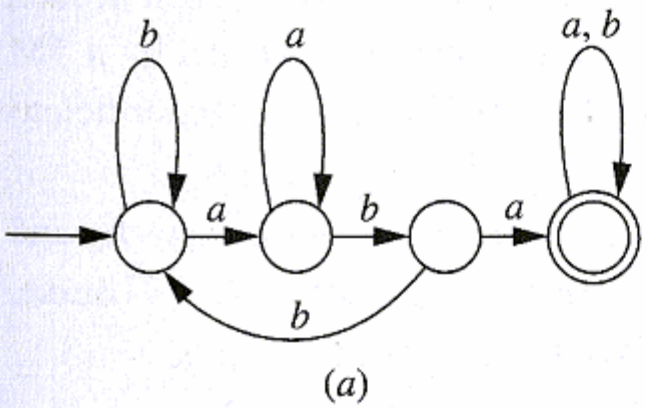




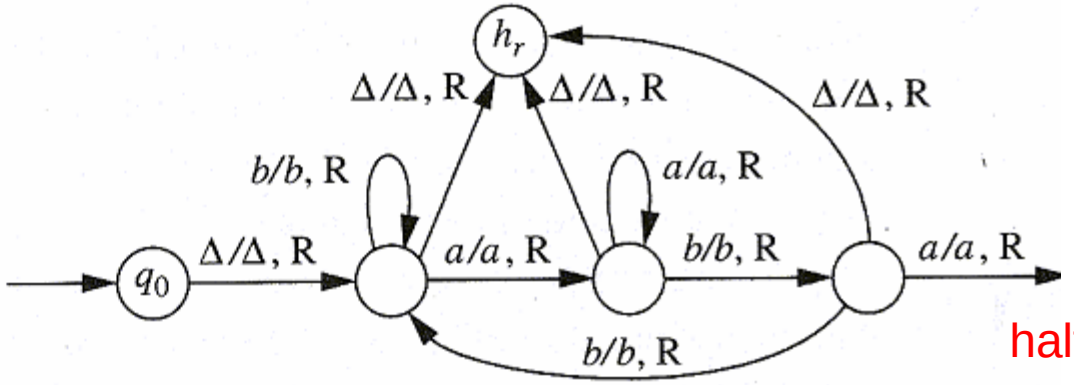
(b,b,R) OR (b/b,R) → (i/p / o/p, Move)

(a)DFA

(b) & (c) TM to accept {a,b}*{aba} {a,b}*

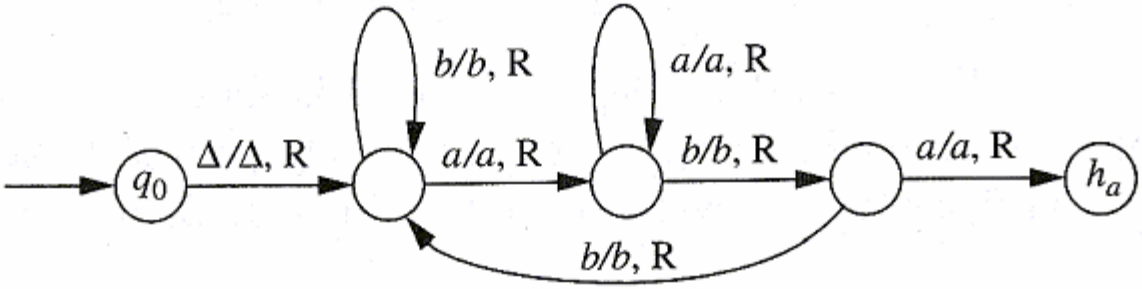


(a)



(b)

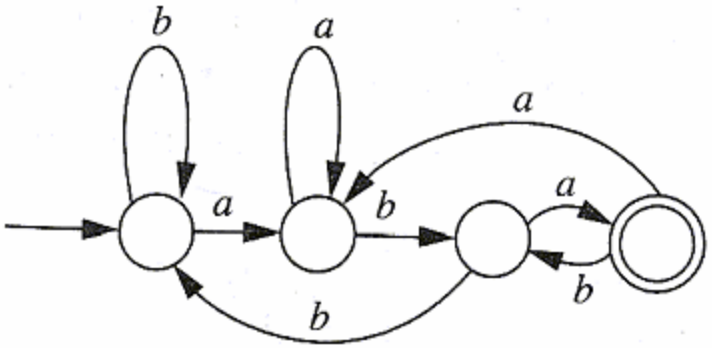
halt



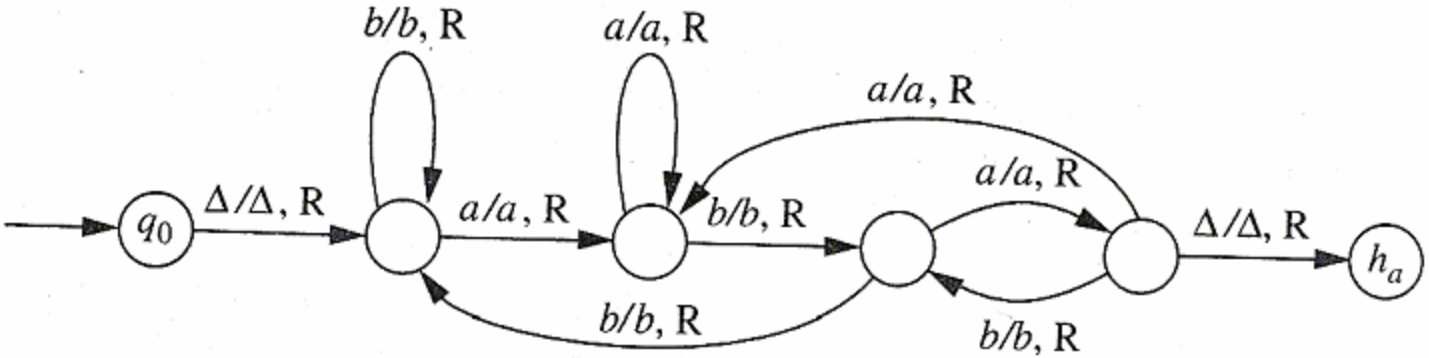
(c)

(a)DFA

(b)TM to accept {a,b}*{aba}



(a)



(b)

Halting Problem of TM

1. **String accept (ha):** $(q_0, \underline{\Delta} x) \vdash_T^* (h_a, y\underline{a}z)$
2. **Delta partial function (hr):** TM enters into undefined configuration
3. **Crash (hr):** Leftmost square with left move
4. **Infinite loop (hr):** Never ending sequence

Combining TM's

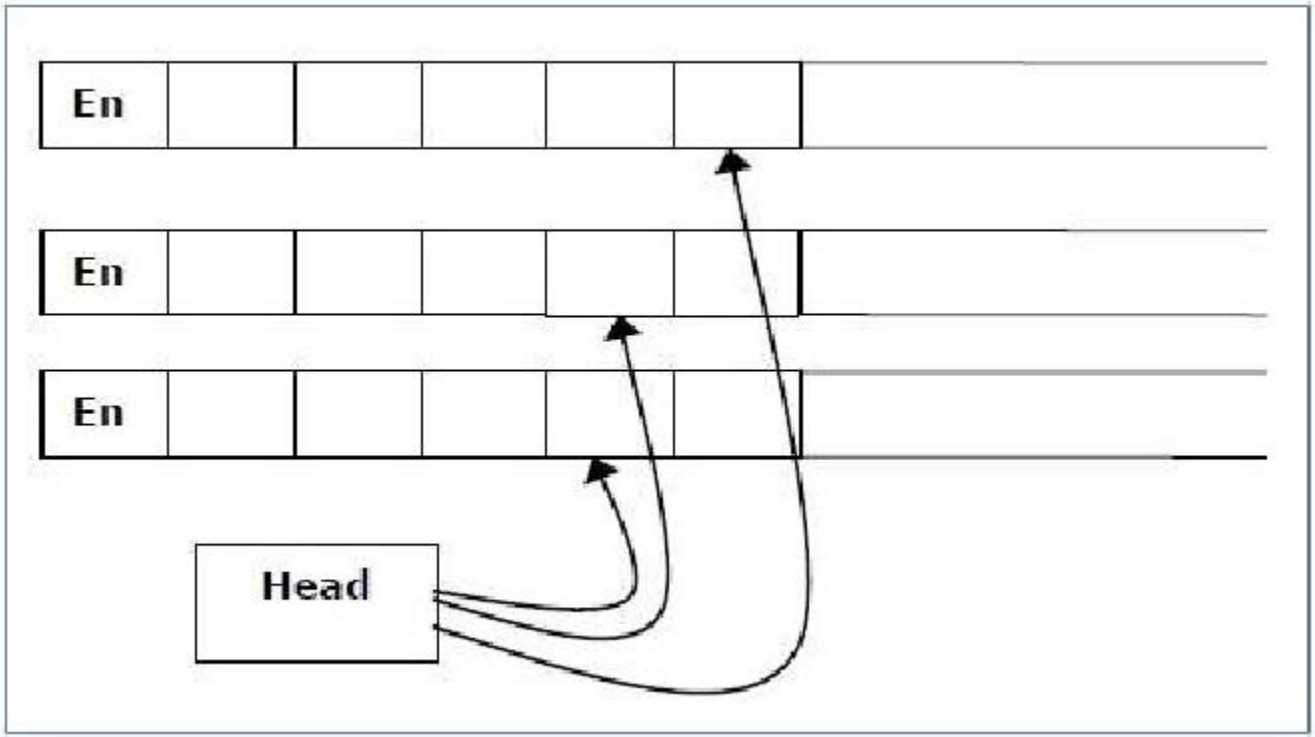
T1 – Q1
T2 – Q2 **Q1 ∩ Q2 = Ø** **T1T2 – Q1 U Q2**

- Starts with initial state of T1;
- with halting problem of T1 starts with the state where T1 halts
- Enters into accept/reject state of T2
- T1T'T2
- Improves power

MODIFICATIONS in TM

- Multi– tape TM
- Multi –track TM
- Non– deterministic TM
- TM with semi—infinite tape

MULTI-TAPE TM



$\delta: Q \times X^k \rightarrow Q \times (X \times \{Left\_shift, Right\_shift, No\_shift\})^k$
where there is **k** number of tapes

Multi-track TM

- Multi-track Turing machines, a specific type of Multi-tape Turing machine, contains multiple tracks but just one tape head reads and writes on all tracks. Here, a single tape head reads n symbols from n tracks at one step.

- **δ** is a relation on states and symbols where,

- $\delta(Q_i, [a_1, a_2, a_3, ...]) = (Q_j, [b_1, b_2, b_3, ...], \text{Left\_shift or Right\_shift})$

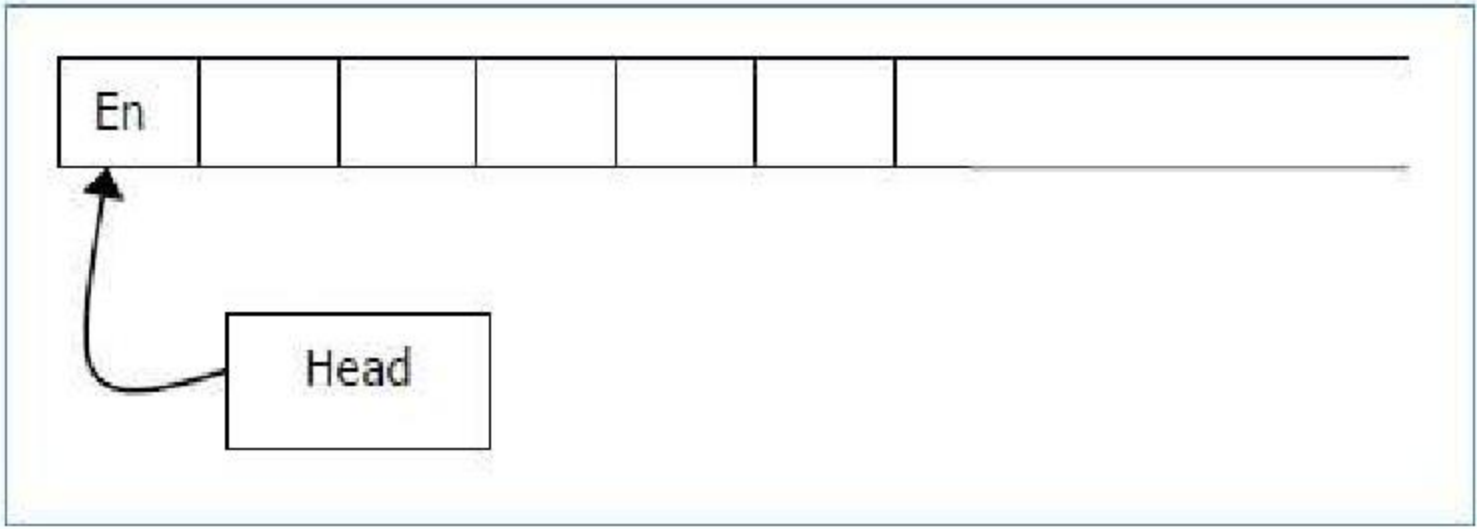
NON-DETERMINISTIC TM

- In a Non-Deterministic Turing Machine, for every state and symbol, there are groups of actions the TM can have. So, the transitions are not unique and not deterministic.
- The computation of a non-deterministic Turing Machine is a tree of configurations that can be reached from the start configuration.
- An input is accepted if there is at least one node of the tree which is an accept configuration, otherwise it is not accepted.
- δ is a transition function :
- $\delta : Q \times X \rightarrow 2^{(Q \times X \times \{Left\_shift, Right\_shift\})}$.

Right hand of transition is a power set.

TM WITH SEMI-INFINITE TAPE

- A Turing Machine with a semi-infinite tape has a left end but no right end.
  (Finite end at left but non finite end at right).
  The left end is limited with an end marker.



ADVANTAGES

Turing machines are similar to finite automata/finite state machines but have the advantage of unlimited memory.

They are capable of simulating common computers; a problem that a common computer can solve (given enough memory) will also be solvable using a TM.

Simplicity of proofs

As a theoretic model, TMs are "simple" in the sense that the current machine state has only constant size. All the information you need in order to determine the next machine state is *one* symbol and *one* (control) state number. The change to the machine state is equally small, adding only the movement of the machine head.

ADVANTAGES

If you're designing some kind of programming language (or functions or anything else that is meant to compute things), then you may want to ensure that it is **Turing-complete** (i.e., capable of computing anything that is computable) by implementing a Turing machine in it.)

TM helps **to classify decidable problems** into classes of Polynomial Hierarchy. Once the problem is found decidable, Then target it with efficient solution. The efficiency been calculated in number of steps, extra space used , length of the code/size of the FSM. TM is able to improve efficiency of the computations.

APPLICATION

The **Church-Turing thesis** claims that any computable problem can be computed by a Turing machine. This means that a computer more powerful than a Turing machine is not necessary to solve computable problems. The idea of Turing completeness is closely related to this. A system is Turing complete if it can compute every Turing computable function. A programming language that is Turing complete is theoretically capable of expressing all tasks accomplishable by computers; nearly all programming languages are Turing complete.

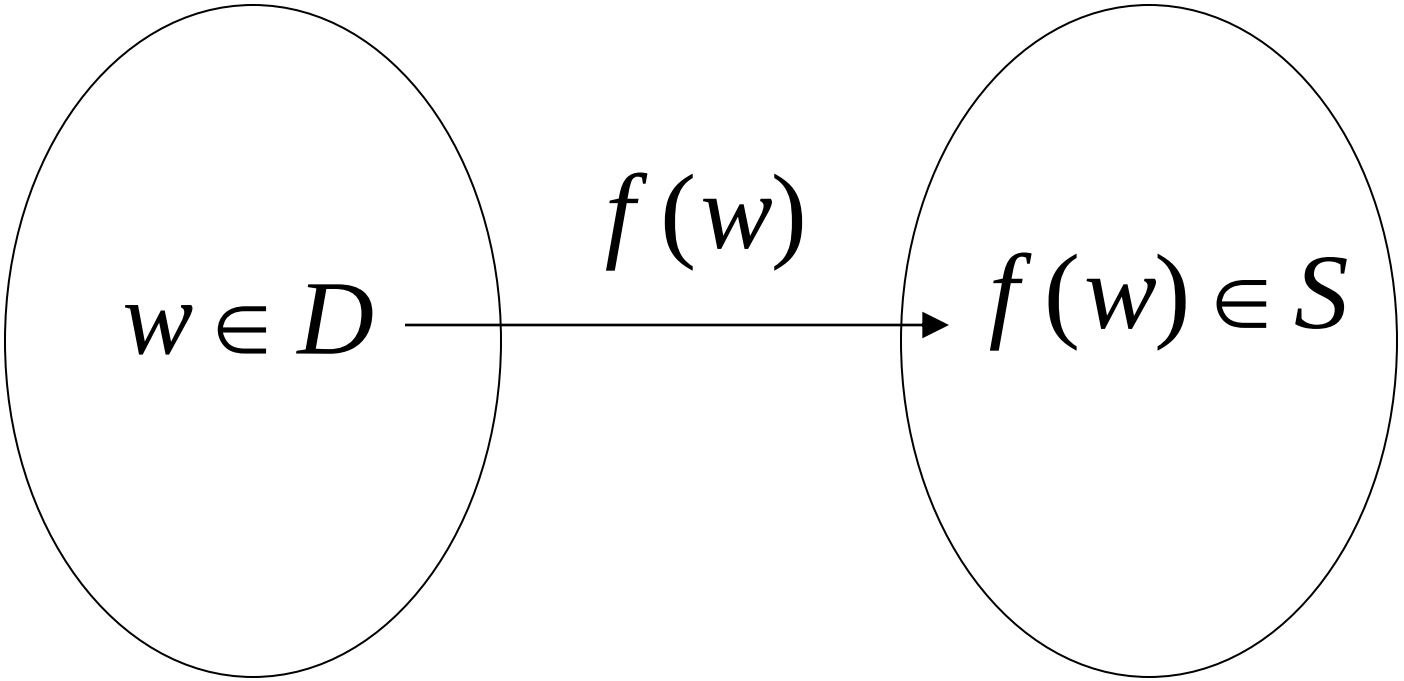
To prove that something is Turing complete, it is sufficient to show that it can simulate some other Turing complete system. Usually, it is easiest to show that a system can simulate A **Universal Turing machine**. A universal Turing machine is a Turing machine that can simulate *any* other Turing machine.

Computing Functions with Turing Machines

A function $f(w)$ has:

Domain: D

Result Region: S



Integer Domain

Decimal: 5

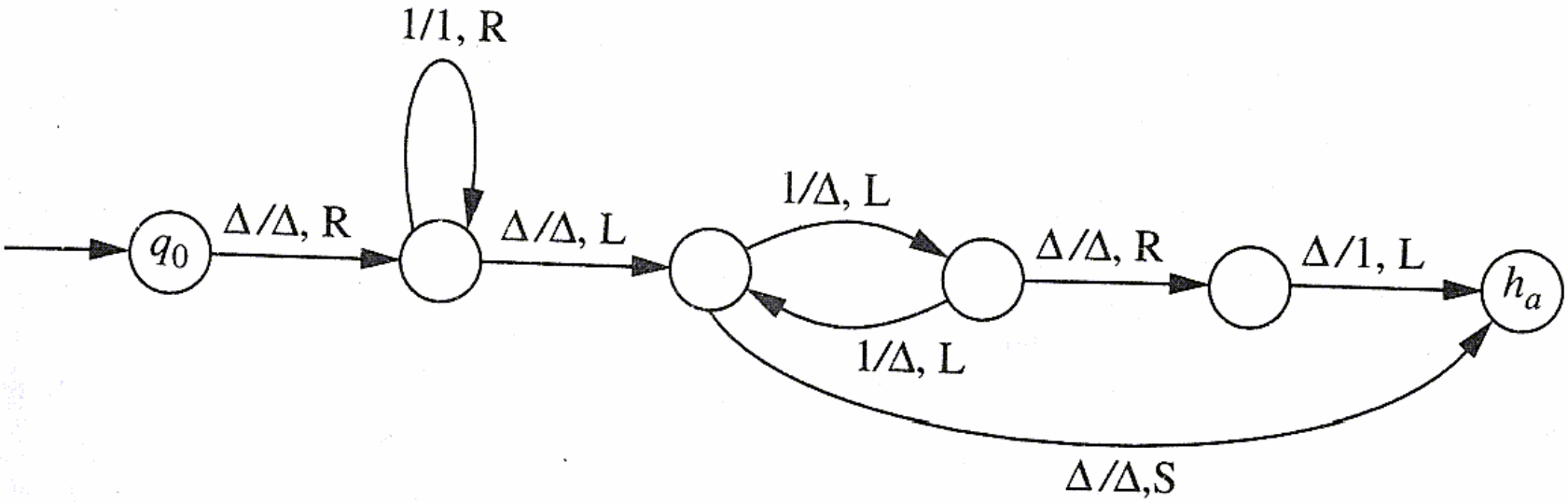
Binary: 101

Unary: 11111

We prefer **unary** representation:

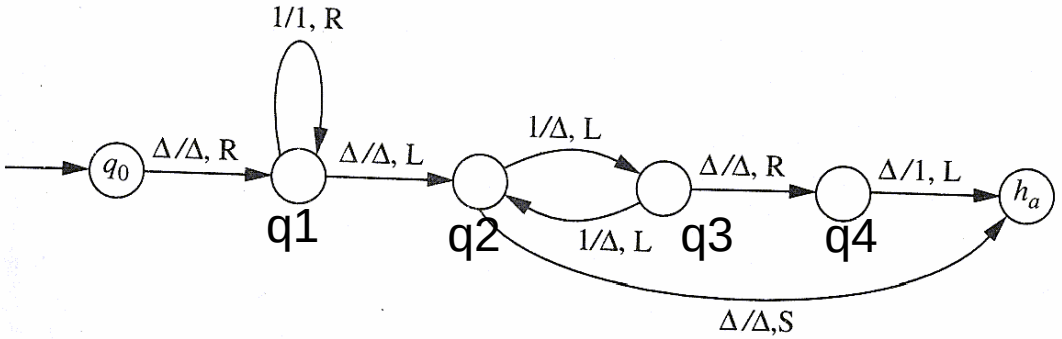
easier to manipulate with Turing machines

TM to compute (n mod 2)



Simulation: ∑ ={1}

W= 5
= 11111



(q0, Δ11111) ⊢ (q1, Δ11111)

⊢* (q1, Δ11111 Δ)

⊢ (q2, Δ11111 Δ)

⊢ (q3, Δ1111 Δ Δ)

⊢ (q2, Δ111 Δ Δ Δ)

⊢ (q3, Δ11 Δ Δ Δ Δ)

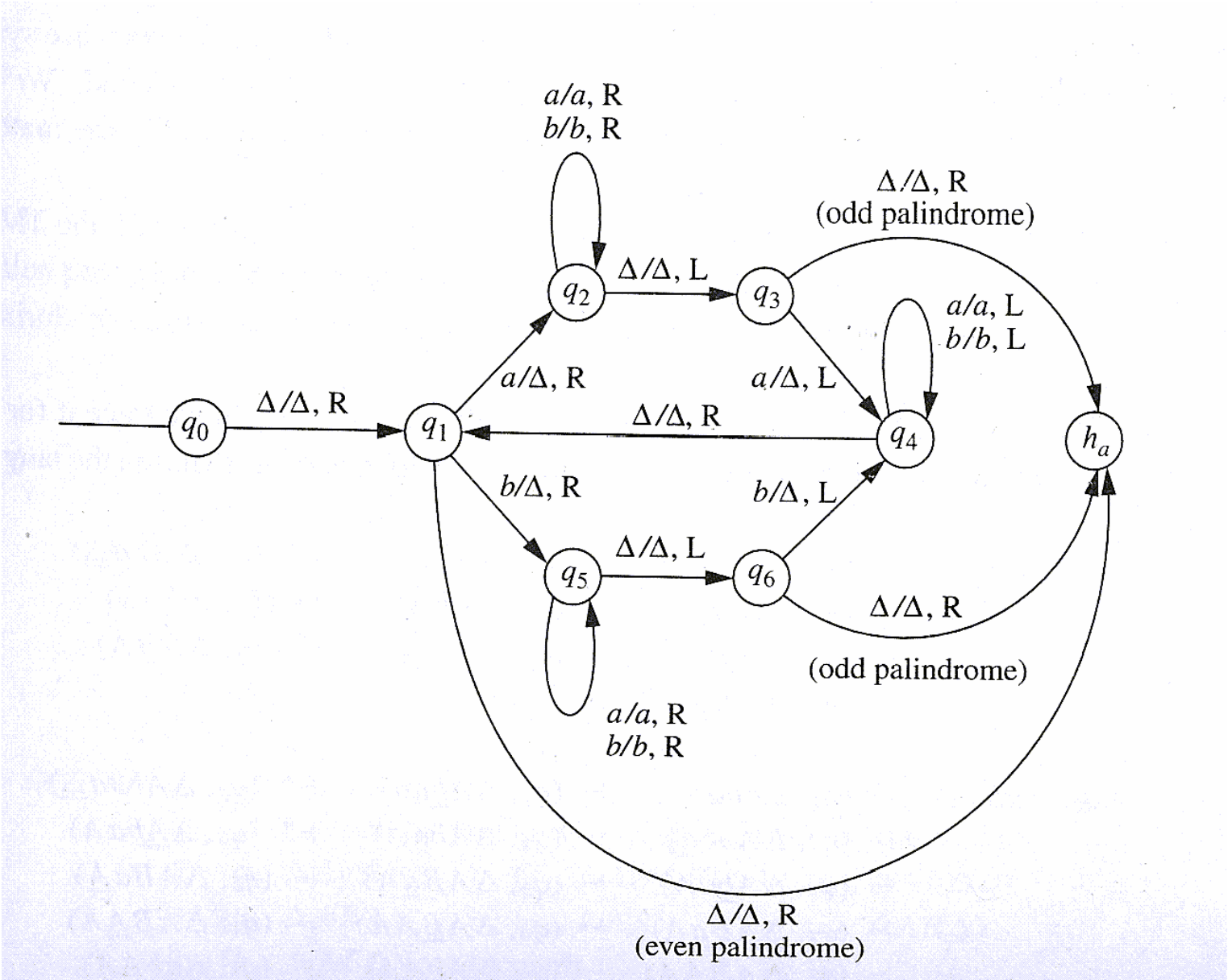
⊢ (q2, Δ1 Δ Δ Δ Δ Δ)

⊢ (q3, Δ Δ Δ Δ Δ Δ Δ)

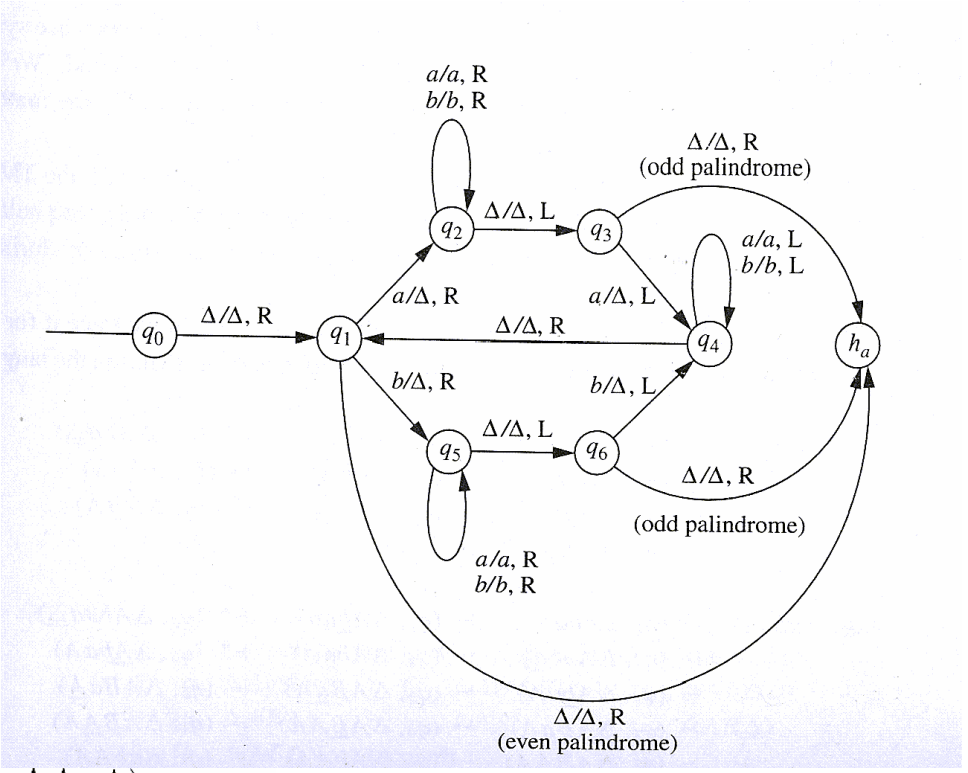
⊢ (q4, Δ Δ Δ Δ Δ Δ Δ)

⊢ (ha, Δ 1Δ Δ Δ Δ)

TM accepting *palindromes* over {a,b}



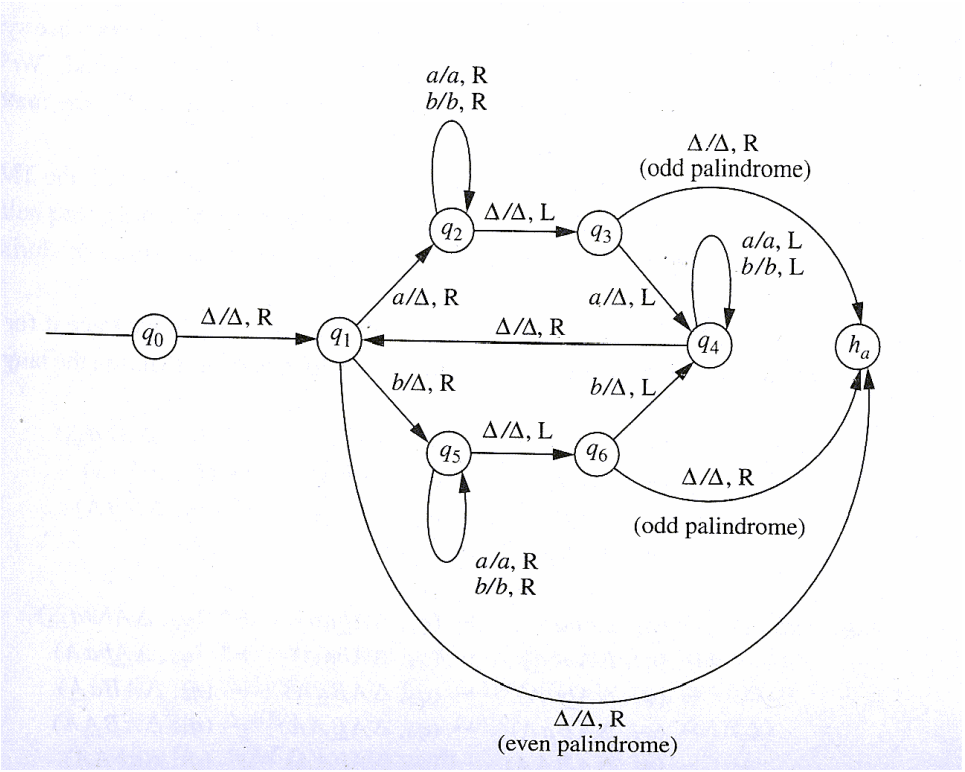
Accept Language



$(q_0, \underline{\Delta}aa) \vdash (q_1, \Delta\underline{a}a) \vdash (q_2, \Delta\Delta\underline{a}) \vdash (q_2, \Delta\Delta a\underline{\Delta})$
$\vdash (q_3, \Delta\Delta\underline{a}) \vdash (q_4, \Delta\Delta\underline{\Delta}) \vdash (q_1, \Delta\Delta\underline{\Delta})$
$\vdash (h_a, \Delta\Delta\Delta\underline{\Delta})$ (accept)

$(q_0, \underline{\Delta}aba) \vdash (q_1, \Delta\underline{a}ba) \vdash (q_2, \Delta\Delta\underline{b}a) \vdash^* (q_2, \Delta\Delta ba\underline{\Delta})$
$\vdash (q_3, \Delta\Delta\underline{b}a) \vdash (q_4, \Delta\Delta\underline{b}) \vdash (q_4, \Delta\underline{\Delta}b)$
$\vdash (q_1, \Delta\Delta\underline{b}) \vdash (q_5, \Delta\Delta\Delta\underline{\Delta}) \vdash (q_6, \Delta\Delta\underline{\Delta})$
$\vdash (h_a, \Delta\Delta\Delta\underline{\Delta})$ (accept)

Reject Language



$$
\begin{aligned}
(q_0, \underline{\Delta}abaa) &\vdash (q_1, \Delta\underline{a}baa) \vdash (q_2, \Delta\Delta\underline{b}aa) \vdash^* (q_2, \Delta\Delta baa\underline{\Delta}) \\
&\vdash (q_3, \Delta\Delta baa) \vdash (q_4, \Delta\Delta b\underline{a}) \vdash^* (q_4, \Delta\underline{\Delta}ba) \\
&\vdash (q_1, \Delta\Delta\underline{b}a) \vdash (q_5, \Delta\Delta\Delta\underline{a} \vdash (q_5, \Delta\Delta\Delta a\underline{\Delta}) \\
&\vdash (q_6, \Delta\Delta\Delta\underline{a}) \vdash (h_r, \Delta\Delta\Delta a\underline{\Delta}) \quad \text{(reject)}
\end{aligned}
$$

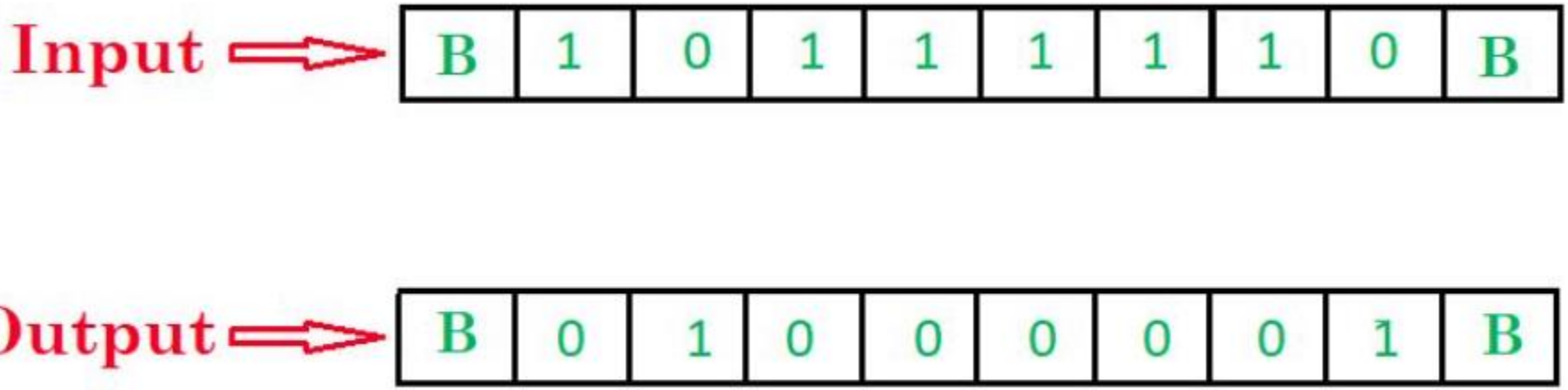
Ex1:
Design TM for successor function ∑ ={1}

(q0, Δ111 Δ) ⊢* (ha, Δ1111Δ)

Ex2:
Design TM for predecessor function ∑ ={1}

(q0, Δ111 Δ) ⊢* (ha, Δ11Δ)

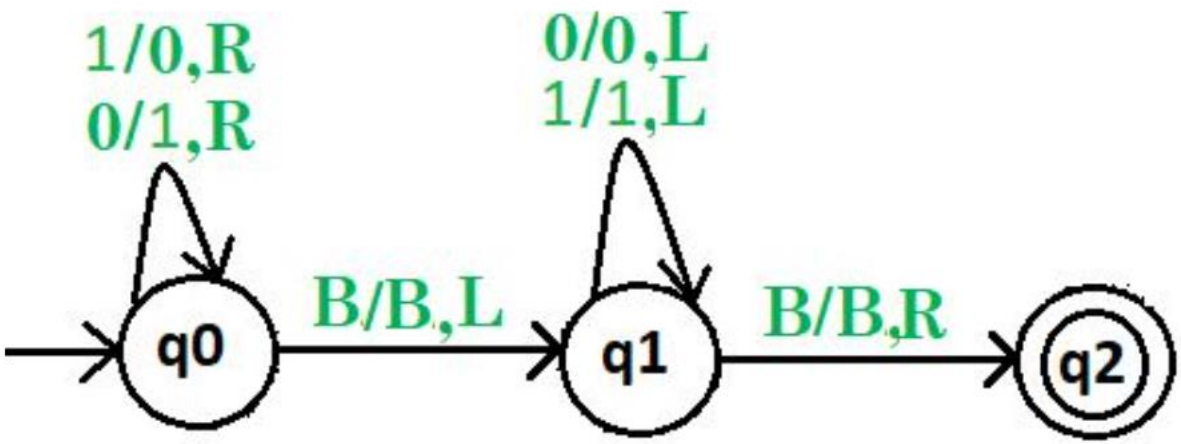
Design TM for 1's complement of a binary number



Design TM for 1's complement of a binary number

Approach:

1. Scanning input string from left to right
2. Converting 1's into 0's
3. Converting 0's into 1's
4. Move the head to the start when BLANK is reached.



- Insert initial blank
- Replace final state with halt_accept

Design TM for 2's complement of a binary number

2's complement of binary numbers can be done by using two approaches.

A) Adding 1's complement+1

B) Traverse bits from right to left, find the 1st 1 bit then reverse all the bits after the 1 bit.

Ex1: Input be 1110010 ➔ Output − 0001110

Ex2:

B	0	1	0	0	0	0	1	0	0

The output is as follows −

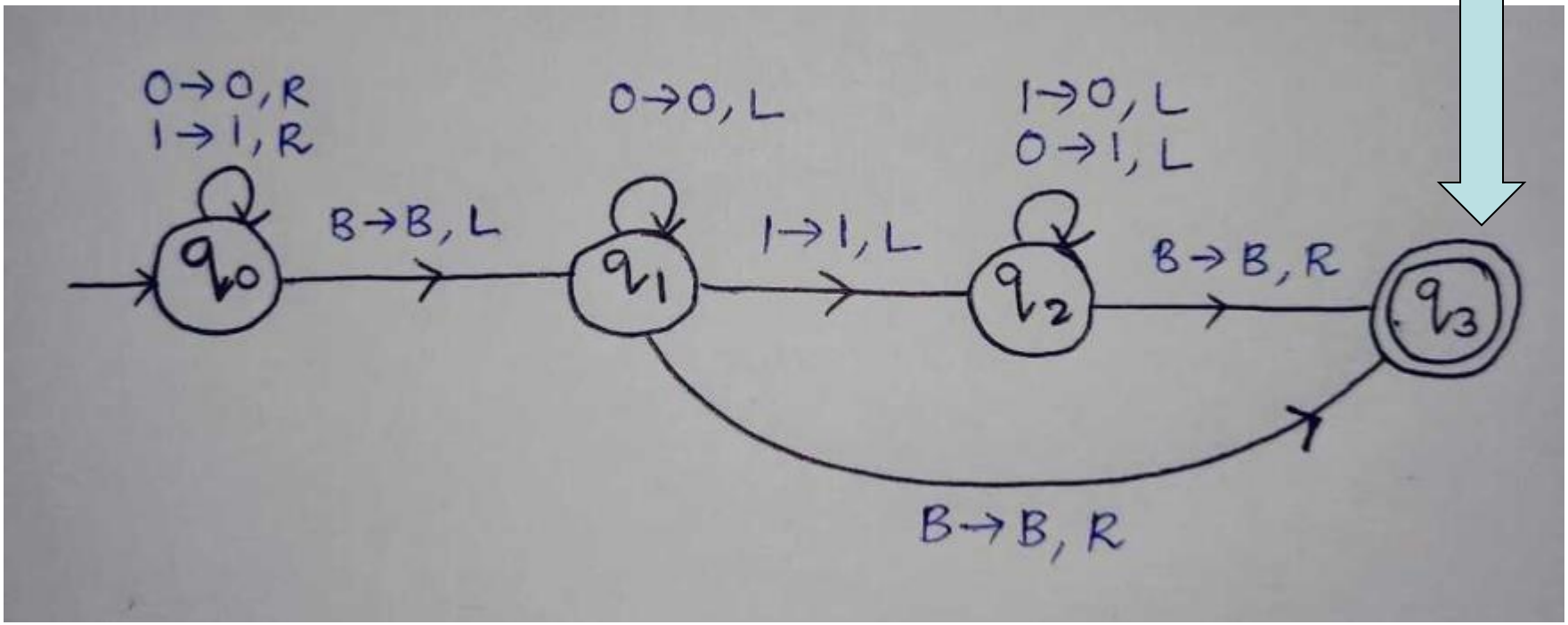
B	1	0	1	1	1	1	1	0	0

Design TM for 2's complement of a binary number

Approach:

1. Scanning input string from right to left
2. Pass all consecutive '0's
3. For first '1' comes, do nothing
4. After that, Converting 1's into 0's and Converting 0's into 1's
5. Stop when BLANK is reached.

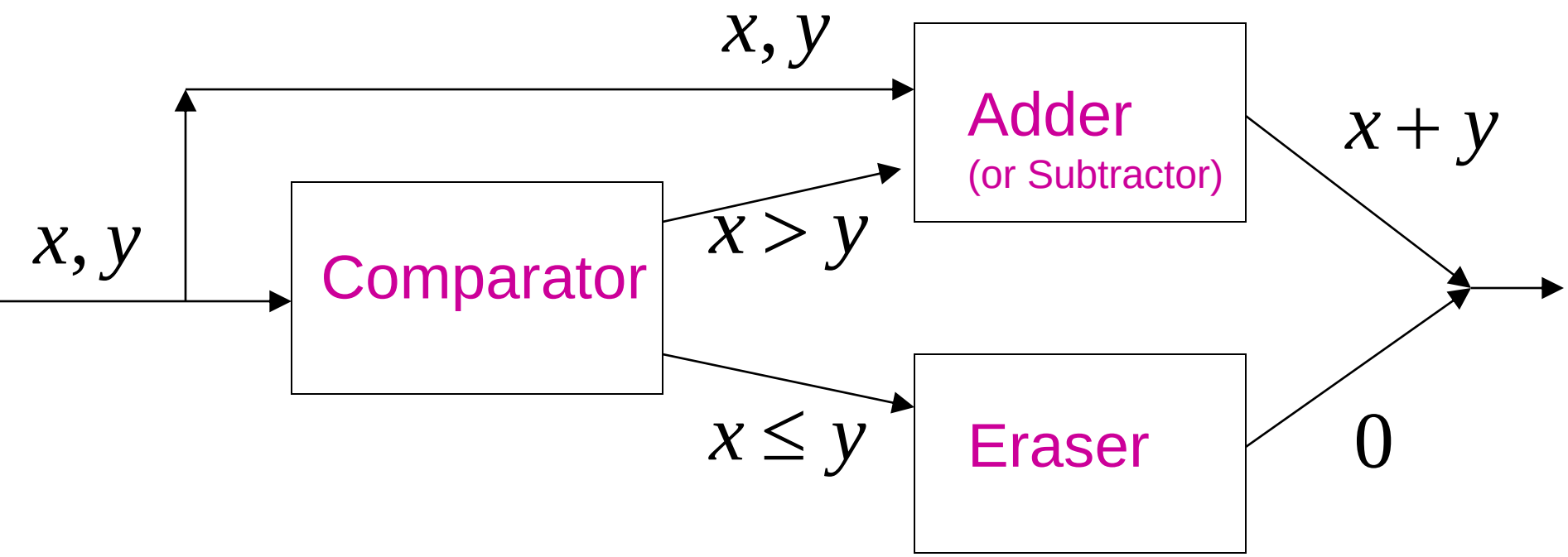
- Replace with halt_accept



Combining Turing Machines

Example:

$$f(x,y)=\begin{cases} x+y & \text{if} \quad x>y \\ 0 & \text{if} \quad x\le y \end{cases}$$



The Universal Turing Machine

Problem : All our machines so far are hardwired.

Question : Can we build a programmable TM that accepts as input:

program *input string*

executes the program, and outputs:

output string

The Universal Turing Machine

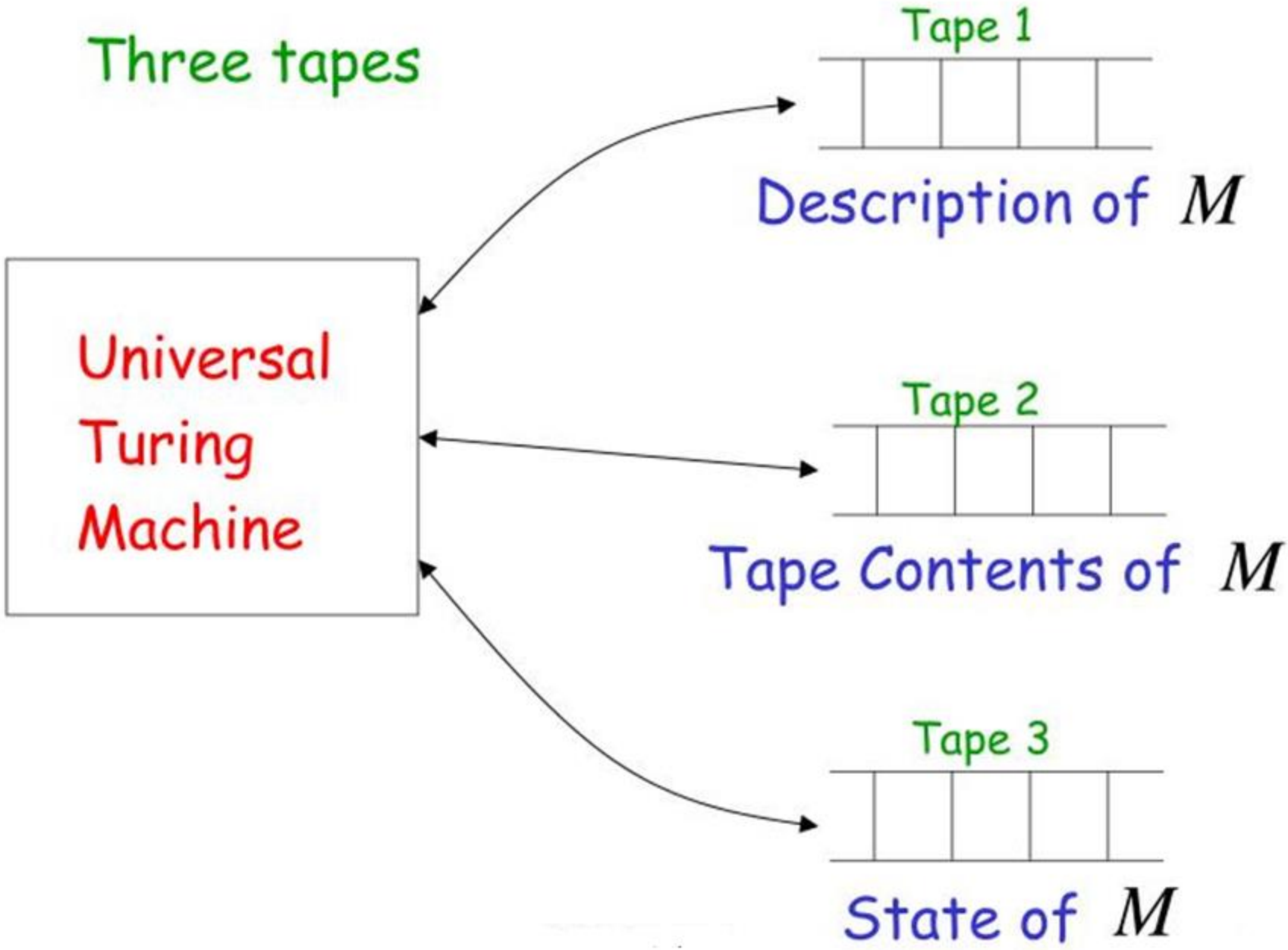
Yes, it's called the ***Universal Turing Machine*** .

To define the Universal Turing Machine *U* we need to:

1. Define an encoding operation for TMs.

2. Describe the operation of *U* given input <*M*, *w*>, the encoding of:

    - a TM *M*, and
    - an input string *w*.


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

Universal TM

- A Universal Turing Machine can simulate any other Turing machine !
- When started on a tape containing the encoding of another Turing machine, call it "T", followed by 't' the input to "T", a UTM produces the same result as "T" would when started on that input.
- When a UTM is given a program (a description of another machine), it makes itself behave as if it were that machine while processing the input.
- A machine "T" working on input 't' is likely to execute far fewer transitions that a UTM simulating T working on t, but for our purposes this fact is irrelevant.
- **The Universal Turing Machine is theoretically feasible.** Constructing it (or simulating it) proves to be daunting challenge.
- One way to think of a UTM is as a programmable computer that in principle can 'solve' any problem.

THANK YOU!…