

Turing Machine

Prof. (Dr.) K.R. Chowdhary

Email: kr.chowdhary@iitj.ac.in

Formerly, Prof. & HOD, Dept. of CSE MBM Engg. College

Sunday 20th July, 2025

K.R. Chowdhary

Theory of Computation

Automata, Formal Languages, Computation and Complexity

Focuses on pedagogy in its writing, that represents a refreshing approach

Ensures comprehensive and enjoyable learning

Undergone a rigorous classroom testing



© 2025

Get 20% off with this code: **SPRAUT**

Available on Springer Nature Link

[link.springer.com/book/
9789819762347](https://link.springer.com/book/9789819762347)

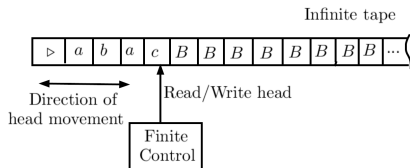
Please note that promotional coupons are only valid for English-language Springer, Apress, and Palgrave Macmillan books & eBooks and are redeemable on link.springer.com only. Titles affected by flood book price laws, forthcoming titles and titles temporarily not available on Springer Nature Link are excluded from promotions, as are reference works, handbooks, encyclopedias, subscriptions, or bulk purchases. The currency in which your order will be invoiced depends on the billing address associated with the payment method used, not necessarily your home currency. Regional VAT/tax may apply. Promotional prices may change due to exchange rates. Promotions are valid for individual customers only. Bookellers, book distributors, and institutions such as libraries and corporations, please visit springernature.com/contact-us. Promotions do not work in combination with other discounts or gift cards.

- Alan Turing was one of the founding fathers of CS.
- His computer model - the Turing Machine (TM) - was inspiration /premonition of the electronic computer which came two decades after the TM model
- Invented the *Turing Test* for in AI
- Legacy: *The Turing Award, eminent award in CS research*

Definition

(Church-Turing Thesis:) TM is ultimate model for computation. Any thing, which is solvable, i.e., has an algorithm, what ever the computation model is used to compute that algorithm, it is ultimately the TM model.

Turing Machine Model for computation



$$M = (Q, \Sigma, \Gamma, \delta, s, H),$$

where, Q is set of states

H is set of Halting states, $H \subseteq Q$

Σ is set of input symbols

Γ is tape alphabet, $\Gamma = \Sigma \cup \{B, \triangleright\}$

δ is transition function (a partial function),

$\delta : (Q - H) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

Definition

Language acceptability by TM:

$$L = \{w \mid w \in \Sigma^*, q_0 w \vdash^* \alpha p \beta, p \in H; \alpha, \beta \in \Gamma^*\}$$

Turing Machine

- TM has a infinite amount of read / write memory. Input is assumed to reside on the tape.
- 1936(Alan M. Turing): Any logical/arithmetic computation, for which complete instructions for carrying out this are supplied, it is always possible to design a TM that can perform this computation.
- TM v/s Human: TM model is based on human

problem solving process using pencil and paper. As we do this, our mental state changes, for every smallest step.

Correspondingly, TM has tape (=paper), R/W head (=pencil), and state (=state of our mind).

- $\{a^n b^n | n \geq 0\}$ v/s $\{a^n b^n c^n | n \geq 0\}$
- Powers of TM: Power is problem solving capability, and not about how fast or slow it can do.

These are Automata, as simple as possible - to define formally, describe and reason about them, as general as possible (any computation can be represented by them).

Definition

(Acceptability by Turing machine:) A string w is accepted by TM M if after being put on the tape with the TM head set to the left-most position, and letting M run, M eventually enters the halting state. In this case w is an element of $L(M)$, the language accepted by M is,

$$L(M) = \{w \mid w \in \Sigma^* \wedge q_0 w \vdash^* \alpha y \beta\},$$

where, y is halting configuration, and $\alpha, \beta \in \Gamma^*$

Turing Machine solves a Problem: Erase entire tape

Consider a TM

$M = (Q, \Sigma, \Gamma, \delta, s, H)$, $Q = \{q_0, q_1\}$,

$\Sigma = \{a\}$, $\Gamma = \{a, B, \triangleright\}$, $s = q_0$, B is blank character, \triangleright is left end marker.

$H = \{q_1\}$

$\delta(q_0, a) = (q_0, B, R)$

$\delta(q_0, B) = (q_1, B, L)$

Let $w = aaaa$

q_0aaaaB

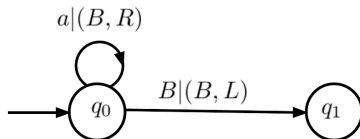
$\vdash Bq_0aaaB$

$\vdash BBq_0aaB$

$\vdash BBBq_0aB$

$\vdash BBBBq_0B$

$\vdash BBBq_1B$



Configuration-1

- A configuration of a TM: $X_1X_2\dots X_{i-1}qX_iX_{i+1}\dots X_n$
 - Current state: (q)
 - Symbols on tape: $X_1\dots X_n$
 - position of RW head: X_i
 - A formal specification of configuration:
 - uqv , where u, v are strings on Σ , and uv is current content on tape, q is current state, and head is at first symbol of v .
- For example, $00101q_5011$ where read head points at 0 (third digit from end) and state is q_5 .

Configuration-2

- For Two configurations:

$uaq_i b v$ and $uq_j a c v$, where $a, b, c \in \Sigma$ and $u, v \in \Sigma^*$

$uaq_i b v \vdash uq_j a c v$ if $\delta(q_i, b) = (q_j, c, L)$

$uaq_i b v \vdash uacq_j v$ if $\delta(q_i, b) = (q_j, c, R)$

- Two special cases:

- The left most cell:

$q_i b v \vdash q_j c v$ for $\delta(q_i, b) = (q_j, c, L)$

$q_i b v \vdash c q_j v$ for $\delta(q_i, b) = (q_j, c, R)$

- On the cell with blank symbol:

uaq_i is equivalent to $uaq_i B$

Example: of language recognition

Design TM to accept: $\{a^n b^n \mid n \geq 0\}$

Let $M = (Q, \Sigma, \Gamma, \delta, s, H)$. The algorithm can be specified as:

1. M replaces left most 'a' by 'X', and then head moves to right until it encounters left most b
2. Replaces this b by Y , and then moves left to find the right most X . Then moves one step right to left most a
3. Repeat Step 2 and 3 in order, i.e., 2, 3, 2, 3, ...
4. When searching for b , if finds a blank character B (i.e., $|a^n| > |b^n|$), then M does not accept w .
5. If a is not found but it finds b , then also M does not accept, (i.e., $|a^n| < |b^n|$).
6. After changing last b to Y , if M finds no more a then it checks that no more b remains. If this is true then $a^n b^n$ is accepted by M i.e., $|a^n| = |b^n|$

Example: of language recognition

$$Q = \{q_0, q_1, q_2, q_3, q_4\}$$

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

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

$$s = q_0$$

$$H = \{q_4, q_0\}$$

- Design TM to accept: $\{a^n b^n \mid n \geq 0\}$

1. $\delta(q_0, a) = (q_1, X, R)$

$\delta(q_1, a) = (q_1, a, R)$; skip through a 's and

$\delta(q_1, Y) = (q_1, Y, R)$; then Y 's

$\delta(q_1, b) = (q_2, Y, L)$

$\delta(q_2, Y) = (q_2, Y, L)$, traverse through Y 's and then

$\delta(q_2, a) = (q_2, a, L)$, traverse a 's

Example: of language recognition...

TM to accept: $\{a^n b^n \mid n \geq 0\}$

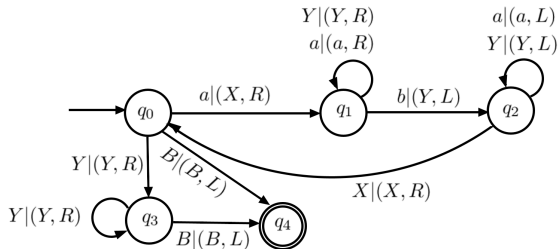


Figure 1: Transition diagram for TM accepting Language $L = \{a^n b^n \mid n \geq 0\}$.

- Move from R to L until X is found and start back:
 $\delta(q_2, X) = (q_0, X, R)$, right most X is found. Now repeat from 1 else from 2.
- 2. $\delta(q_0, Y) = (q_3, Y, R)$, scan through Y's
 $\delta(q_3, B) = (q_4, B, L)$, accept w , and halt.

Transition Table for $\{a^n b^n \mid n \geq 0\}$

Current State	Tape Symbol a	Tape Symbol b	Tape Symbol B	Tape Symbol X	Tape Symbol Y
q_0	(q_1, X, R)	-	(q_4, B, L)	-	(q_3, Y, R)
q_1	(q_1, a, R)	(q_2, Y, L)	-	-	(q_1, Y, R)
q_2	(q_2, a, L)	-	-	(q_0, X, R)	(q_2, Y, L)
q_3	-	-	(q_4, B, L)	-	(q_3, Y, R)
q_4	-	-	-	-	-

Example: of language recognition Dry Run

- TM to accept: $\{a^n b^n \mid n \geq 0\}$, Let $w = aabb$

$q_0 a a b b B \vdash X q_1 a b b B \vdash X a q_1 b b B \vdash X q_2 a Y b B \vdash X q_2 a B b B$
 $\vdash q_2 X a Y b B \vdash X q_0 a Y b B \vdash X X q_1 Y b B \vdash X X Y q_1 b B$
 $\vdash X X q_2 Y Y B \vdash X q_2 X Y Y B \vdash X X q_0 Y Y B \vdash X X Y q_3 Y B$
 $\vdash X X Y Y q_3 B \vdash X X Y q_4 Y B$ (accept the input)

Total number of transitions for $|w| = n$ are: $n/2$ forward and $n/2$ in backward, in each to and fro round, i.e., n . Since, in each trip, two symbols are marked, therefore, there will be total $n/2$ trips, making total transitions: $n \times n/2 = n^2/2$.
Time complexity, $O(n^2/2) = O(n^2)$, which is polynomial (P) time complexity.

Two-tape Turing Machine

Example: Construct TM to recognize $L = \{ww^R \mid w \in \{a, b\}^*\}$.

Approach: We can traverse w ; each time if a/b is found in begin, replace it by B and also, replace a/b at end by B .

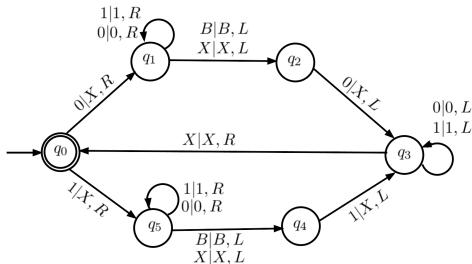


Figure 2: TM to recognize ww^R strings.

Time complexity: Let $|w| = n$. Total of trips = $n/2$. Number of transitions in successive trips are: $n + (n - 2) + (n - 4) + \dots + 4 + 2 = n^2/2 - n/2$ in even palindromes, and $n^2/2 - n/2 + 2$ in odd palindromes, which is $O(n^2)$ is polynomial time complexity.

Acceptors v/s Deciders

- Let M is TM .
- Three possibilities occur on a given input w :
- M eventually enters q_{acc} and therefore halts and **accepts**.
 $w \in L(M)$
- M eventually enters q_{rej} or crashes somewhere. M **rejects** w ,
i.e., $w \notin L(M)$
- M never halts its computation and is caught up in an infinite loop. In this case w is neither accepted nor rejected. However, any string not explicitly accepted is considered to be outside the accepted language. $w \notin L(M)$
- **Decider**: M never enters infinite loop(A **recursive language**).

Fermats last theorem / Church's Thesis

- Consider Fermat's equation(last theorem): $x^n + y^n = z^n$, where n is a positive integer

Question: Given an n , does this equation have a solution in integers?

We could create a Turing machine T which would search for solutions when given an input n by working through a list of integers.

Then $T(n)$ would halt when $n=1$ (having found a solution, such as $x=1, y=1$ and $z=2$, and when $n=2$ (having found a solution such as $x=3, y=4$ and $z=5$

But T would not halt for any other input

- *Church's Thesis*: Any reasonable attempt to model mathematically computer algorithms and their performance, is bound to end up with a model of computation and associated time cost, that is equivalent to Turing machines within a polynomial.

Theorem

Algorithms are countably infinite.

Proof.

- An algorithm consists of sequence of alphabet strings, and every alphabet can be encoded into an integer.
- Let there be 1000 alphabets, and encoded as, e.g., $a = 501, b = 502, c = \dots, ! = 700, \dots$
- Let an algorithm text be $baaa!$
- \therefore it can be encoded as 502501501501700. Similarly every algorithm can be encoded as integer (of course an integer number may be very-very large). If two encoding are identical, then the corresponding algorithms are identical.
- All these integers form the set: \mathbb{N} .
- This concludes that algorithms are countable and infinite.



Undecidability

Theorem: Show that there are infinitely many problems which are not solvable.

Primality Proof: • Let a black box with i/p and i/p , producing a mapping such that for i/p sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, ..., output is 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, (If i/p is prime o/p is 1, else 0). Concatenate outputs and insert "0" before each, producing 0.0110101000101.... Let this problem be p_1 . so, problem p_1 maps to 0.0110101000101..., with each

prime number as an instance of the problem.

- Every problem can thus have a mapping to a binary fraction.
- The Set of all these fractions, representing the *bijection* with problems, is set of real numbers: $[0,1]$, is *uncountably infinite*.
- With countably infinite algorithms (\mathbb{N}), and uncountably infinite problems (\mathbb{R}), we conclude: there exists infinitely many problems with no algorithms.

Language acceptability by TM

- A language accepted by TM is *recursive (R)* if corresponding TM always halts, i.e. on accepting it as well as rejecting.
- A language accepted by TM is *recursively enumerated(RE)* if the corresponding TM halts only on accepting it. $R \subset RE$.
- Church's Lambda-calculus paradigm was proved equivalent to TM.
- A Turing machine (TM) is similar to a finite automaton with an unlimited and unrestricted memory, and having writing capability.

A Thinking Machine:

- It was hard for the ancients to believe that any algorithm could be carried out on such a simple device. For us, it's much easier to believe, especially if you have programmed in assembly!
- A Turing machine can do everything that a real computer can do.
- But, a Turing machine cannot solve certain classes of problems.



Chowdhary, K.R. (2025). Turing Machine and Computability.
In: Theory of Computation. Springer, Singapore.
https://doi.org/10.1007/978-981-97-6234-7_9