Turing Machines

Interested in:

what are effective or mechanical methods for "computing" functions?

What can be computed by humans?

What are models for such computation?

Turing's Thesis: a Turing machine can do anything that can be described as "rule of thumb" or "purely mechanical."

Church's Thesis: a function of positive integers is effectively calculable only is lamda-definable (equivalently recursive).

Equivalent models

(1) λ-definable (Church) functions

(2) partial μ-recursive functions (Herbrand-Gödel, etc)

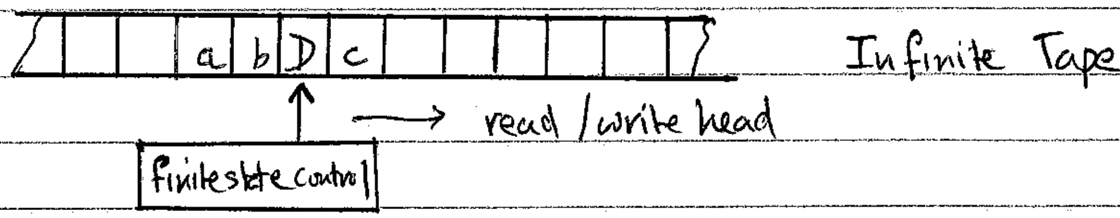
(3) Logical computing machines (Turing)

⋮

other models also exist

Most useful in many ways turned out to be Turing machines

Computability by a Turing machine is the accepted definition of a procedure or effective computation.



$T = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$

Q : finite set of states

Σ : finite set of inputs $\Sigma \subseteq \Gamma - \{\square\}$

Γ : finite set of symbols (tape alphabet; includes $\square$)

$\square$: special symbol

q_0 : initial state $q_0 \in Q$

F : final states $F \subseteq Q$

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

the transition function

"Given a state and a symbol read by read/write head, describes next state and replacement symbol and the read/write head moves either left or right on the tape"

Operations of a T.M.

Assume that T.M. halts if if no move is possible. (that is δ is not defined for a (state, tape symbol))

Starts from an initial configuration with a finite set of non-blank symbols on the ~~t~~ tape.
Continues until it halts or keeps moving forever.

The transition function

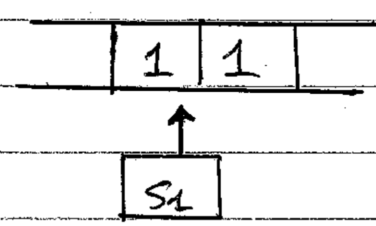
Typically listed as five-tuples (quintuples)
(current state; current symbol; next state; next symbol; move)

Example

"Computing a function"

$Q = \{s_1, s_2, s_3, s_4, s_5\}$ $\Gamma = \{\not b, 1, 2\}$ ($\not b$ → blank symbol.)
$\Sigma = \{1\}$ $q_0 = s_1$

$(s_1, 1, s_1, 1, R)$
$s_1, \not b, s_2, 2, L)$
s_2 $\not b$ s_3 $\not b$ R
s_2 1 s_2 1 L
s_2 2 s_2 2 L
s_3 1 s_4 $\not b$ R
s_3 2 s_5 $\not b$ R
s_4 $\not b$ s_5 1 R
s_4 1 s_4 1 R
s_4 2 s_4 2 R
s_5 $\not b$ s_2 1 L



In this example, initial configuration is a positive number in unary notation.

Computes $f : I \rightarrow I$
$f(n) = 2n$

Instantaneous Descriptions (ID)

$\alpha_1 \, q \, \alpha_2$

or

$$\underbrace{a_1 a_2 \cdots a_{k-1}}_{\alpha_1} \, q \, \underbrace{a_k a_{k+1} \cdots a_n}_{\alpha_2}$$

A	□	B	a_k	□	C
			q		

$A \square B \, q \, a_k \square C$ is ID.

α_1 : tape to the left of the r/w head. (until all blanks)
α_2 : current symbol being read, followed by tape to the right of r/w head. (until all blanks)

A move from one ID (configuration) to the next is denoted by $\vdash$.
Thus, if $\delta(q, A) = (q', B, R)$
then the following move is possible.

$$a_1 a_2 C \, q \, A D a_3 a_4 \vdash a_1 a_2 C B q' D a_3 a_4.$$

$\vdash^*$ $\quad \vdash^+$ $\quad \vdash^k_M$ standard notation.

The sequence of configurations leading to a halt state from the initial configuration is termed a computation.

If the TM does not halt from some configuration we sometimes write $\alpha_1 q \alpha_2 \vdash^* \infty$.

General convention: No moves from final state;
ie it halts in a final state.

Language accepted by a TM.

All words in Σ^* that cause M to enter a final state when placed on the tape with initial state q_0 and scanning leftmost cell of word (or $\square$ for λ).

Formally:
Given a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$, then the language accepted by M is

$$L(M) = \{ w \mid w \in \Sigma^* \text{ and } q_0 w \vdash^* \alpha_1 p \alpha_2 \text{ for } p \in F \text{ and } \alpha_1, \alpha_2 \in \Gamma^* \}$$

Example

TM to accept $\{ 0^n 1^n \mid n \geq 1 \}$

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

$\Sigma = \{ 0, 1 \}$

$\Gamma = \{ 0, 1, X, Y, B \}$ $\quad \square = B$, $\quad F = \{q_4\}$

	δ
replace leftmost 0 by an X, go to state q_1	$q_0, 0 \; ; \; q_1, X, R$
0's exhausted, check for any more 1's	$q_0, Y \; ; \; q_3, Y, R$
move right over 0's	$q_1, 0 \; ; \; q_1, 0, R$
found a 1, replace by Y search left in state q_2	$q_1, 1 \; ; \; q_2, Y, L$
skip right over Y looking for 1	$q_1, Y \; ; \; q_1, Y, R$
move left over 0's	$q_2, 0 \; ; \; q_2, 0, L$
rightmost X found, start over	$q_2, X \; ; \; q_0, X, R$
move left over Y's	$q_2, Y \; ; \; q_2, Y, L$
skip over Y's looking for 1.	$q_3, Y \; ; \; q_3, Y, R$
no more 1's, accept input string.	$q_3, B \; ; \; q_4, B, R$

Try 000111 ; 0101 ; 00111 ; 0001 ;

Recursively Enumerable sets and Recursive Sets.

Def.

A language L is recursively enumerable (r.e.) if there exists some Turing machine TM whose language is L.
Note: the Turing machine may not halt for every input.

Def.

A language L is recursive if there exists some Turing machine TM whose language is L and that halts for every input.

Note: If a language is r.e. there is in general no algorithm to determine membership. (ie. is $x \in L$).

Turing Machines as computing functions

(1) functions of k arguments.

$\underbrace{I \times I \times I \cdots \times I}_{k} \longrightarrow I.$ I is the set of non-negative integers.

(2) represent integers in unary 111 (3)
11111 (5)

separating characters for functions of k arguments.
$S\,11\,S\,111\,S\,11111\,S$ (input for $f(2,3,5)$)

(3) Number of 1's on tape at the end of the computation is the value of what it computes. $\boxed{11\,\not b\,\not b\,111\,\not b\,111} \longleftrightarrow 8$

④ A function computing by a TM
$$f(x_1, x_2, \ldots, x_n)$$
is called a <u>partial</u> re<u>cursive</u> function

Note: the function may not be defined for some arguments (it is a <u>partial</u> function)

Note: if the function is defined for all arguments (it is a total function) it is termed a total recursive function. These functions correspond to Turing machines that <u>halt</u> for <u>every</u> input.

<u>Example</u>

A Turing Machine to compute $x+y$ (ie., $f(x_1, x_2) = x_1 + x_2$).
Assume the input as a function of two arguments is separated by an S.

| B | 1 | 1 | 1 | 1 | S | 1 | 1 | 1 | B | B | B | · |

↑ starting config.

$$(\{q_0, q_1, q_2, q_3\}, \{0,1\}, \{S, 1, B\}, \delta, q_0, B, \phi)$$

$(q_0, 1 \;;\; q_0, 1, R)$
$(q_0, S \;;\; q_1, 1, R)$
$(q_1, 1 \;;\; q_1, 1, R)$
$(q_1, B \;;\; q_2, B, L)$
$(q_2, 1 \;;\; q_3, B, R)$

Higher Level Tools for T.M.'s

① **Storage in Finite Control**

Can make the finite control a vector as long as it is still finite. eg. (q_i, C_j)

Counting the # of A's in the input mod. 5.

Need 5 "count states" C_0, C_1, C_2, C_3, C_4.

② **Multiple tracks**

view the tape alphabet as an m-tuple.

Example $m = 3$

track 1	¢	1	0	1	1	\$	b̸	b̸	
" 2	b̸	-	.	.		. .		b̸	
" 3	b̸	.	.	.		–	.	b̸	

view the input as on track 1.

Example

Test if a number is a prime. (written in binary).

Solution: Use 3 tracks.

① Write a 2 in binary on track 2 (How?)

② Copy track 1 to track 3 (easy)

③ Subtract track 2 from track 3 with result on track 3. (hard but doable)

④ Compare to see if track 3 is less than track 2 else repeat ③

⑤ If remainder is $0 \to$ not prime.

⑥ else increase track 2 number by 1 & try again from ③.

unless if track 2 # = track 3 # then halt, # is prime.

③ <u>Checking off symbols.</u>

<u>Example</u> TM for:
$L = \{ wcw \mid w \in \{a,b\}^+ \}$

a a b c a a b

$$Q = \{ [q_i, s_j], \quad s_j = a, b, B \}$$
$$\Sigma = \{ [X_i, B], \quad X_i = a, b, c \}$$
$$\Gamma = \{ [Z_i, Y_j], \quad Z_i = a, b, c, B; \; Y_j = B, \checkmark \}$$

① start state is $[q_1, B]$, blank symbol is $[B,B]$
$\delta([q_1, B], [d, B]) = [q_2, d], [d, \checkmark], R$ for $d = a, b$
"check scanned symbol; move right with symbol in finite state control"

② $\delta([q_2, d], [e, B]) = [q_2, d][e, B], R$ $\quad d = a, b; \; e = a,$
"move right over unchecked symbols looking for c.

③ $\delta([q_2, d], [c, B]) = [q_3, d], [c, B], R$
" found the c"

④ move right over "checked symbols"

⑤ $\delta[(q_3, d], [d, B]) = [q_4, d], [d, \checkmark], L]$

⋮

---

④ Subroutines & other higher-level tools

Equivalent Models for Turing Machines

1. Allow Turing machine not to move read / write head. moves: $\{L, R, S\}$ $S \equiv$ stay.

How can this be proven?
Answer: By simulation.

Lemma:
Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two "classes" of Turing Machines. The classes are equivalent if for every $M_1 \in \mathcal{C}_1$, there exists an M_2 in $\mathcal{C}_2$ such that $L(M_1) = L(M_2)$ and for each M_2' in $\mathcal{C}_2$ there exists an $M_1' \in \mathcal{C}_1$ such that $L(M_2') = L(M_1')$

Proof of stay-machine class equivalence by "simulation"
for M_1 in standard class, $\hat{M}_1$ where each move is the same for $\hat{M}_1 \in$ StayClass is clearly $L(M_1) = L(\hat{M}_1)$.
Consider $M_2' \in$ stay class. Consider a stay-move:

$$\delta_2'(q, a) = (p, b, S)$$

Define a sequence of two moves in the standard class as follows:

$$\delta_1'(q, a) = (p', b, R)$$
$$\delta_1'(p', x) = (p, x, L)$$

where $x \in \Gamma$ and p' is a non-final state for ever $p \in Q$.

Essentially M_1' <u>simulates</u> moves of M_2', and accepts if and only if M_2' accepts. Thus

$$L(M_1') = L(M_2')$$

2. Machine must change state and may either move or write but not both

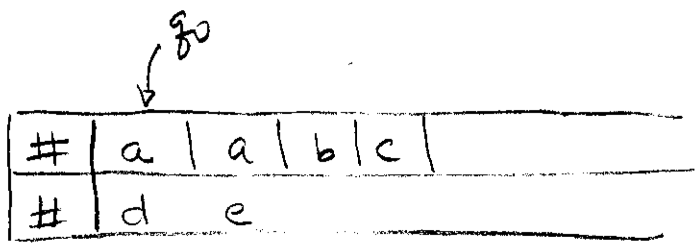
3. Machine must move and may either change state or write but not both.

4. ... etc.

5. Turing machine with 1 way infinite tape $\equiv$ 2 way infinite tape.

Several ways to prove this:

(a) Use two tracks on 1 way tape



Normal Right moves or Left moves on "upper" tape

$$\delta(q_i, [a, x]) = (q_j, [b, x], R \text{ or } L)$$

for $\delta_{\text{standard}}(q_i, a) = (q_j, b, R \text{ or } L)$

$$\delta(q_i, [\#, \#]) = (p_i, [\#, \#], R)$$

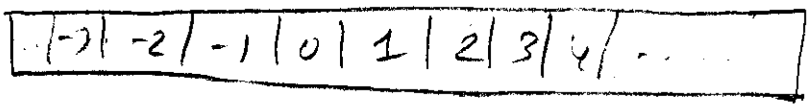
and

$$\delta(p_i, [x, a]) = (p_j, [x, b], L \text{ or } R)$$

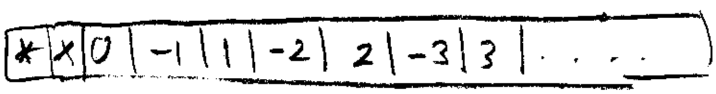
for $\delta_{\text{standard}}(q_i, a) = (q_j, b, R \text{ or } L)$ on "lower tape"

Note: introduced a set or "new" states $p_1 \ldots p_n$.

(b)



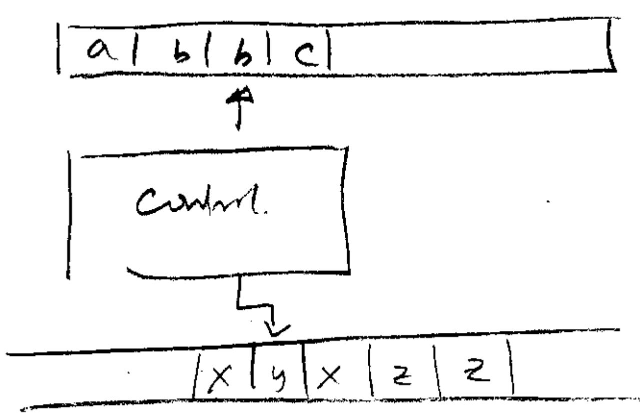
two way tape



one way tape

Basically 1-way tape machine "skips" a cell going right or left
when hit star it changes mode, moves to -1 cell
and then follows analogue moves.

6. "Off-Line" Turing machine



Read only input file

infinite tape.

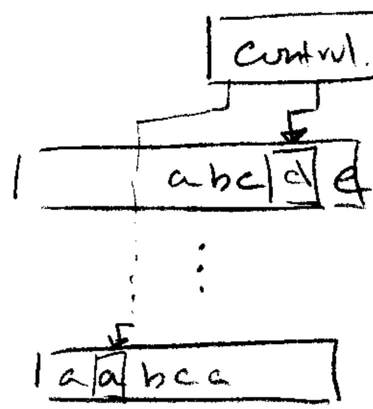
<u>Proof</u> Use 4 track, 1 pair for input file tape & position
1 pair for infinite tape, & position

....		a	b	b	c		
	0	0	0	1	0		
			X	Y	x	z	z
			0	1	0	0	

Note above represents configuration of "off-line" machine.

(7) Multi-tape Turing Machines

Finite control plus k tapes with k independent r/w heads.



tape 1

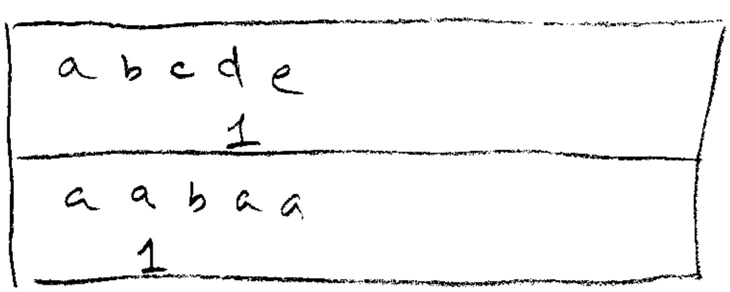
tape k

Transition function is:

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

note each r/w head can move independently.

Multi-tape Turing machines are equivalent to standard Turing machines.



standard machine

Sweep left to right capturing the input under each head
(know there are k r/w heads and can keep track of
to left & # to right of current position
then determine the move & sweep right to left making moves

Nondeterministic Turing Machines

Basic idea is that a deterministic Turing machine can simulate a non-deterministic machine.

$$\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R\}}$$

Each (q, a) has a possible finite set of next moves.
$\therefore \exists$ a maximum # of choices, say r for any (q, a) pair.

Use a 3 tape machine.

① | a b a a b c | input tape

② | 1 2 5 3 4 5 | sequence generator. $r = 5$

③ | | actual working tape.

(a) generate sequences with each digit $\leq r$ in a systematic manner

1
2
⋮
5
1 1
1 2
1 5
2 1
2 2
2 5
⋮
5 5
1 1 1 etc

(b) simulate Nondet machine by using sequence to simulate a specific choice at each step.

A Universal Turing Machine

(1) Encoding of a Turing machine 0,1 s.

Let $Q = \{q_1, \ldots, q_n\}$ $\quad q_1 \xrightarrow{\text{encode}} 1$, $q_2 \to 11 \cdots q_n \to \underbrace{1111}_{n}$

$\Gamma = \{a_1, \ldots a_m\}$ $\quad a_1 \xrightarrow{\text{encode}} 1$, $a_2 \to 11 \cdots a_m \to \underbrace{11111}_{m}$

L, R : 1, 11

Special assumption q_1 is initial state.

q_2 is single final state

a_1 is blank symbol.

$\therefore$ We need only encode

δ. separate components by 0.

$$\delta(q_3, a_3) \to (q_1, a_2, R)$$

encode as

$$111\,0\,111\,0\,1\,0\,11\,0\,11\,0$$

$\therefore$ complete transition function is a sequence of such quintuples, ending say by 00.

This is called the

Description of M.

| 1110 1110 . . . . . | 1st tape

Description of M

Current internal state of M can be put on another tape

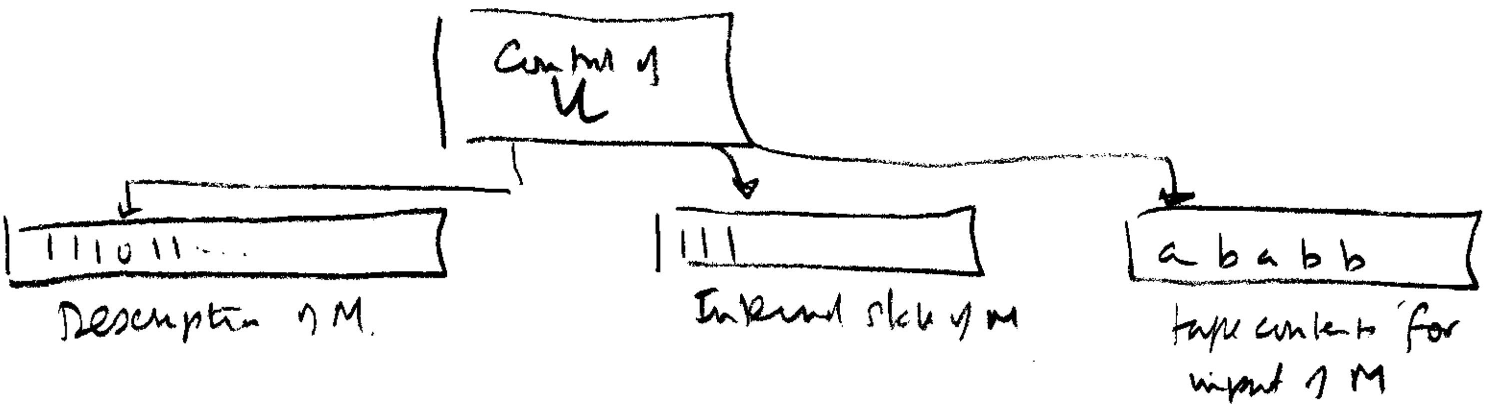
| 1110 | 2nd tape

internal state of M

| | 3rd tape

tape contents of M

A universal Turing machine U, can simulate the behavior of any Turing machine M on w by using the description of M & the input w can simulating the moves of M



Theorem

The set of all Turing machines is countable.

Enumerate Turing Machines

Hierarchy of Languages & Automata

Def. A language L is recursively enumerable if there exists a Turing Machine that accepts the language L.

Def A language L is recursive if the exists a Turing machine that accepts L and halts on every input (in Σ^*).

Lemma There exist enumeration procedures for r.e. and recursive languages.
Proof Outline: generate all strings in some canonical order & then check for acceptance. (have to be careful for r.e. sets to do it in terms of increasing # of moves.

Theorem
There exists a language that is not recursively enumerable
Proof Outline: Set of all languages is uncountable.

Theorem
There exists a recursively enumerable language whose complement is not r.e.
Proof Outline: diagonalization.

Let us enumerate all T.Ms. Let $D(M_i) = x_i$

M_1	x_1	(descriptions)	$L(M_1)$	is $x_1 \in L(M_1)$?
M_2	x_2	$\vdots$	$L(M_2)$	is $x_2 \in M(M_2)$?
M_3	x_3		$\vdots$	$\vdots$
$\vdots$	$\vdots$			

Theorem: There exists a recursively enumerable language whose complement is not r.e.

Proof Outline. (Diagonalization). Enumerate all Turing Machines M_i, $(i = 1, 2, 3, \ldots)$. Let x_i be the description $D(M_i) = x_i$.

M_1	$D(M_1) = x_1$	$L(M_1)$	Is $x_1 \in L(M_1)$?
M_2	$D(M_2) = x_2$	$L(M_2)$	Is $x_2 \in L(M_2)$?
M_3	$D(M_3) = x_3$	$L(M_3)$	Is $x_3 \in L(M_3)$?
$\vdots$			

Let $L = \{ x_i \mid x_i \in L(M_i) \}$
Let $\bar{L} = \{ x_i \mid x_i \notin L(M_i) \}$

Claim $\bar{L}$ is not r.e. Proof by contradiction
If $\bar{L}$ is r.e., then for some k, $L(M_k) = \bar{L}$.
Consider x_k.

$\therefore$ If $x_k \in L(M_k) \Rightarrow x_k \in \bar{L} \Rightarrow x_k \notin L(M_i)$ def of $\bar{L}$

If $x_k \notin L(M_k) \Rightarrow x_k \notin \bar{L} \Rightarrow x_k \in L \Rightarrow$ ~~$x_k \in M_k$~~.
$x_k \in L(M_k)$.

Thus claim that $\bar{L}$ is r.e. must be false.

Note L is r.e. Why? A universal turing machine can simulate M_i on input x_i using $D(x_i)$ for all i. If $x_i \in L(M_i)$ it will halt and say yes.

<u>Theorem</u>

There exists an R.E. language whose complement is not recursive. (use previous theorem.)

Chomsky Hierarchy

Grammars		Languages		Automata
Regular grammars	$\longleftrightarrow$	regular expressions	$\longleftrightarrow$	f.s.a. det or nondet
		det. context free		dpda
context free		context-free		npda
context sensitive		context sensitive		l.b.a
		recursive		
Phrase structure type 0.		recursively enumerable		T.M. (det nondet

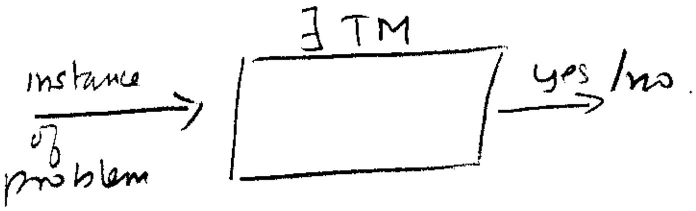
Undecidability & the Halting Problem

By a problem we generally mean a class of problems, a specific instance of which is specified by specific parametric values.

Is a given c.f.g. ambiguous?
· Does TM k accept any string?
Will TM k halt on word m

Answer should be yes/no.

Def
A problem is decidable if its language is recursive; ie



otherwise, the problem is undecidable.
ie, no algorithm will take an arbitrary instance of the problem and reply yes/no for sure.

The Halting Problem is undecidable.

Does Turing machine M accept w as input?

Proof Outline

i Words w \ TM j	1	2	3	4	5	...
1						
2						
3						
4						
5						

entry $(i,j) = 1 \iff TM_j$ accepts w_i.

$L_d = \{ w_i \mid (i,i) = 0 \}$ that is,
$w_i \in L_d \iff$ ith word not accepted by i^{th} Turing machine

Claim 1 L_d is not r.e. ; $\overline{L_d}$ is not recursive
$\overline{L_d} = \{ w_i \mid M_i \text{ accepts } w_i \}$

Claim 2
Let a string (M,w) consist of the description of a TM followed by the word w (with appropriate separators)
Consider $L_u = \{ \langle M, w \rangle \mid M \text{ accepts } w \}$
L_u is r.e. (proof - universal TM).

Claim 3
L_u is not recursive.

Proof:
L_u recursive $\Rightarrow \overline{L_d}$ recursive.
(run a restricted L_u on $\langle M_i, w_i \rangle$

Halting Problem

Claim 1 L_d is not r.e.

Suppose L_d is r.e. $\therefore \exists$ an M_j such that $L(M_j) = L_d$ or M_j accept L_d. Now consider w_j.

If $w_j \in L(M_j)$ then $w_j \in L_d \Rightarrow w_j$ is not accepted by $M_j \Rightarrow w_j \notin L(M_j)$. contradiction

If $w_j \notin L(M_j)$ then $(M_j, w_j) = 0$

$\Rightarrow w_j \in L_d \Rightarrow w_j \in L(M_j) = L_d$. contradiction

Thus L_d is not r.e.

Also, $\overline{L_d}$ is <u>not</u> recursive.

Claim 2. L_u is r.e. easy, simulate w on M.

Claim 3. L_u is <u>not</u> recursive. (ie not decidable).

Suppose L_u is recursive.

$\therefore$ for all $\langle M, w \rangle$, L_u will halt yes / no.

$\therefore$ for $\langle M_i, w_i \rangle$, L_u will halt yes/no

$\therefore$ for all w_i, if M_i accepts w_i is decidable,

or $\{ w_i \mid M_i \text{ accepts } w_i \}$ is recursive.

or $\overline{L_d}$ is recursive. This contradicts claim 1.

- The study of the efficiency of algorithms
  Time - complexity
  Space - complexity

Model : Turing Machine
Problem Size : Some integer n characterizing problem size
Analysis : Behavior as n increases

Time complexity : $T(n)$
Space complexity : $S(n)$

$T(n)$ is the maximum number of steps the TM takes on an input of size n.
(worst case performance as a function of problem size)

Big O notation
The function f is of order g,
$$f = O(g)$$
if $\exists$ c and n_0 such that
$$f(n) \le c \cdot g(n) \quad \text{for } n \ge n_0.$$

Examples
Naive sorting is $O(n^2)$
Efficient sorting is $O(n \log n)$
Sorting must take at least $O(n)$ simply to read input.

A language L is a member of $DTIME(T(n))$ if there exists a multitape DTM that decides L in time $O(T(n))$. (DTM = Deterministic TM).

A Language L is a member of $NTIME(T(n))$ if there exist a multitape NTM that decides L in time $O(T(n))$. (NTM = Non-deterministic TM).

Note :

$$DTIME(T(n)) \subseteq NTIME(T(n))$$
$$DTIME(n^k) \subseteq DTIME(n^{k+1})$$
$$L_{Reg} \subseteq DTIME(n)$$

$$P = \bigcup_{i \geq 1} DTIME(n^i).$$

All languages that are decided in polynomial time, by a deterministic TM.

$$NP = \bigcup_{i \geq 1} NTIME(n^i).$$

All languages that are decided in polynomial time by a non-deterministic TM.

$$P \subseteq NP$$

However is

$$P = NP \ ?$$