THE POST CORRESPONDENCE PROBLEM

- The Post Correspondence Problem (PCP) was first introduced by Emil Post in 1946
- The problem over an alphabet belongs to a class of
- yes/no problems and is stated as follows:
- Consider the two lists *x* =(X1 .. Xn*),*
- *Y* = (Y1 ... Yn*)* of nonempty strings over an alphabet {0,1}
- The PCP is to determine whether or not there exist $i_1, \ldots, i_m$, where $1 \le i_j \le n$, such that

$$x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}$$

Note: The indices i_j's need not be distinct and m may be greater than n. Also, if there exists a solution to PCP, there exist infinitely many solutions.

Does the PCP with two lists $x = (b,\ bab^3,\ ba)$ and $y = (b^3,\ ba,\ a)$ have a solution?

We have to determine whether or not there exists a sequence of substrings of x such that the string formed by this sequence and the string formed by the sequence of corresponding substrings of y are identical. The required sequence is given by $i_1 = 2$, $i_2 = 1$, $i_3 = 1$, $i_4 = 3$, i.e. (2, 1, 1,3), and $m = 4$. The corresponding strings are

$$\begin{array}{cccccccccc} \boxed{bab^3} & \boxed{b} & \boxed{b} & \boxed{ba} & = & \boxed{ba} & \boxed{b^3} & \boxed{b^3} & \boxed{a} \\ x_2 & x_1 & x_1 & x_3 & & y_2 & y_1 & y_1 & y_3 \end{array}$$

Thus the PCP has a solution.

Bab6a=bab6a 2,1,1,3

Prove that PCP with two lists $x = (01, 1, 1)$, $y = (01^2, 10, 1^1)$ has no solution.

For each substring $x_i \in x$ and $y_i \in y$, we have $|x_i| < |y_i|$ for all i. Hence the string generated by a sequence of substrings of x is shorter than the string generated by the sequence of corresponding substrings of y. Therefore, the PCP has no solution.

Note: If the first substring used in PCP is always x_1 and y_1, then the PCP is known as the *Modified Post Correspondence Problem.*

Prove that the PCP with $\{(01, 011), (1, 10), (1, 11)\}$ has no solution. (Here, $x_1 = 01$, $x_2 = 1$, $x_3 = 1$, $y_1 = 011$, $y_2 = 10$, $y_3 = 11$.)

Show that the PCP with $S = \{(0, 10), (1^20, 0^3), (0^21, 10)\}$ has no solution. [*Hint:* No pair has common nonempty initial substring.]

Some More Examples on PCP

- Obtain the solution for the following system of posts correspondence problem, X = {100, 0, 1}, Y = {1, 100, 00}.
- **Solution:** The solution is 1, 3, 1, 1, 3, 2, 2. The string is
- X1X3X1X1X3X2X2 = 100 + 1 + 100 + 100 + 1 + 0 + 0 = 1001100100100
  Y1Y3Y1Y1Y3Y2Y2 = 1 + 00 + 1 + 1 + 00 + 100 + 100 = 1001100100100

Some More Examples on PCP

Does PCP with two lists x = (b, a, aba, bb) and y = (ba, ba, ab, b) have a solution?

Solution: Now we have to find out such a sequence that strings formed by x and y are identical. Such a sequence is 1, 2, 1, 3, 3
list

1	2	1	3	3	4		1	2	1	3	3	4
b	a	b	aba	aba	bb	=	ba	ba	ba	ab	ab	b

Decidability and Undecidability

Recursive Language:

- A language 'L' is said to be recursive if there exists a Turing machine which will accept all the strings in 'L' and reject all the strings not in 'L'.
- The Turing machine will halt every time and give an answer (accepted or rejected) for each and every string input.

Recursively Enumerable Language:

- A language 'L' is said to be a recursively enumerable language if there exists a Turing machine which will accept (and therefore halt) for all the input strings which are in 'L'.
- But may or may not halt for all input strings which are not in 'L'.

The recursive languages are a proper subset of the recursively enumerable languages.

Decidable Language:

A language 'L' is decidable if it is a recursive language. All decidable languages are recursive languages and vice-versa.

Partially Decidable Language:

A language 'L' is partially decidable if 'L' is a recursively enumerable language.

Undecidable Language:

- A language is undecidable if it is not decidable.
- An undecidable language may sometimes be partially decidable but not decidable.
- If a language is not even partially decidable, then there exists no Turing machine for that language

Recursive Language	TM will always Halt
Recursively Enumerable Language	TM will halt sometimes & may not halt sometimes
Decidable Language	Recursive Language
Partially Decidable Language	Recursively Enumerable Language
UNDECIDABLE	No TM for that language

Properties of Recursively Languages

Theorem 9.3: If L is a recursive language, so is $\overline{L}$.

PROOF: Let $L = L(M)$ for some TM M that always halts. We construct a TM $\overline{M}$ such that $\overline{L} = L(\overline{M})$ by the construction suggested in Fig. 9.3. That is, $\overline{M}$ behaves just like M. However, M is modified as follows to create $\overline{M}$:

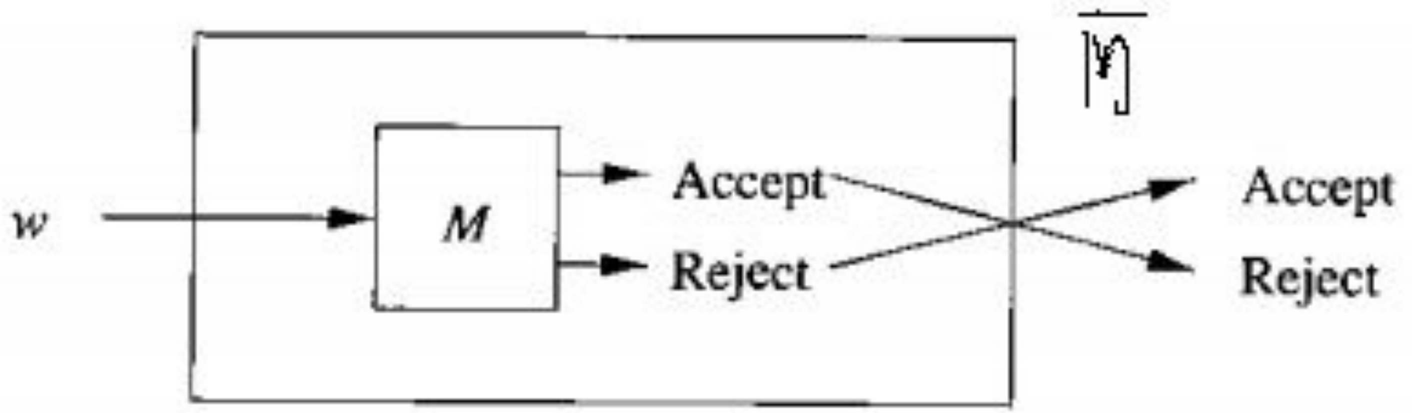
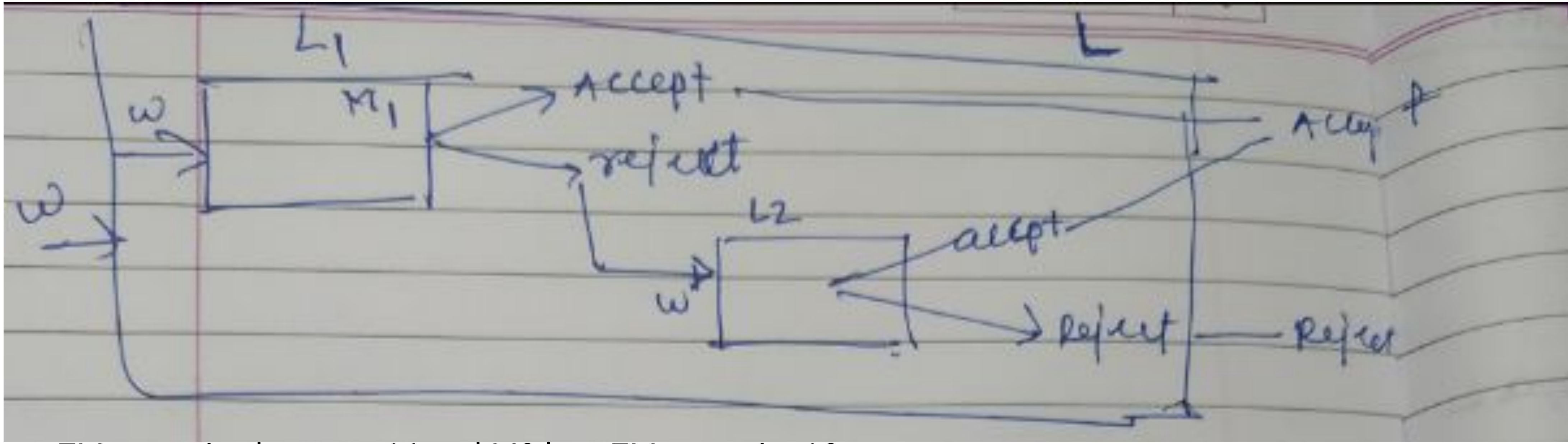


Figure 9.3: Construction of a TM accepting the complement of a recursive language

Since M is guaranteed to halt, we know that $\overline{M}$ is also guaranteed to halt. Moreover, $\overline{M}$ accepts exactly those strings that M does not accept. Thus $\overline{M}$ accepts $\overline{L}$. $\square$

Properties of Recursively Languages

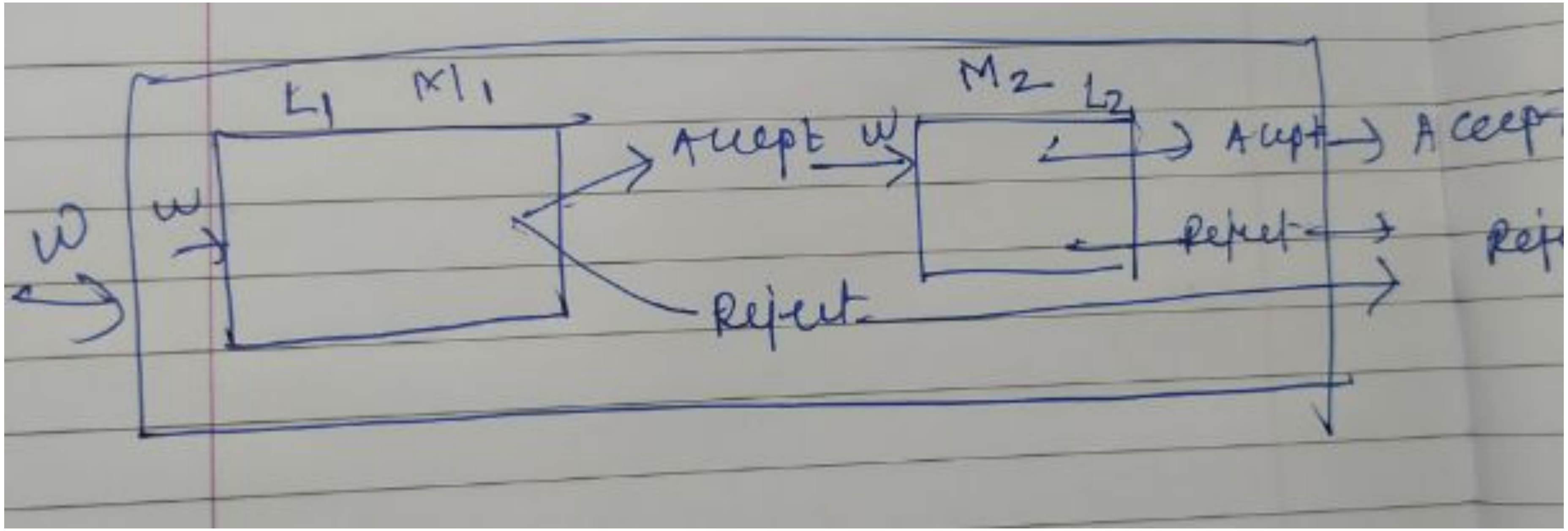
Union of 2 Recursive languages is also Recursive



Let M1 be a TM accepting language L1 and M2 be a TM accepting L2.
M be TM accepting L1 union L2.
M accepts if either M1 or M2 accepts
M rejects if both M1 and M2 rejects.

Properties of Recursively Languages

Intersection of 2 Recursive languages is also Recursive



Let M1 be a TM accepting language L1 and M2 be a TM accepting L2.
M be TM accepting L1 intersection L2.
M accepts if both M1 or M2 accepts
M rejects if either M1 and M2 rejects.

Properties of Recursively Enumerable (RE) Languages

Theorem 9.4: If both a language L and its complement are RE, then L is recursive. Note that then by Theorem 9.3, $\overline{L}$ is recursive as well.

PROOF: The proof is suggested by Fig. 9.4. Let $L = L(M_1)$ and $\overline{L} = L(M_2)$. Both M_1 and M_2 are simulated in parallel by a TM M. We can make M a two-tape TM, and then convert it to a one-tape TM, to make the simulation easy and obvious. One tape of M simulates the tape of M_1, while the other tape of M simulates the tape of M_2. The states of M_1 and M_2 are each components of the state of M.

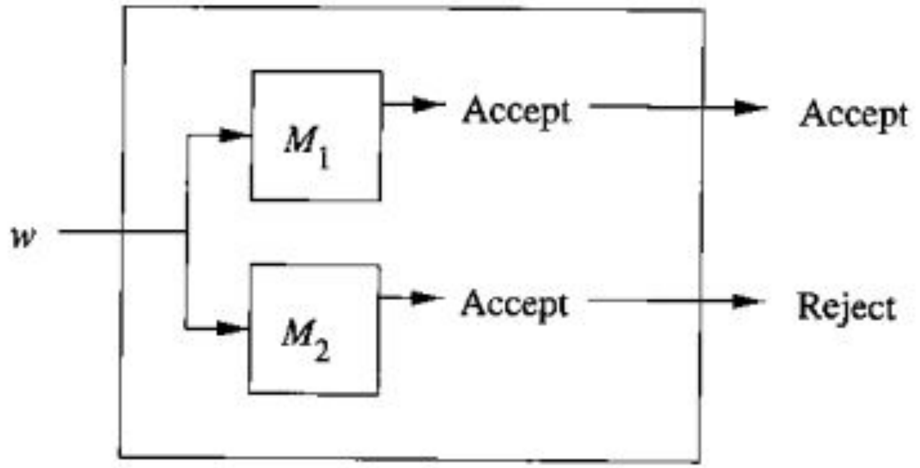
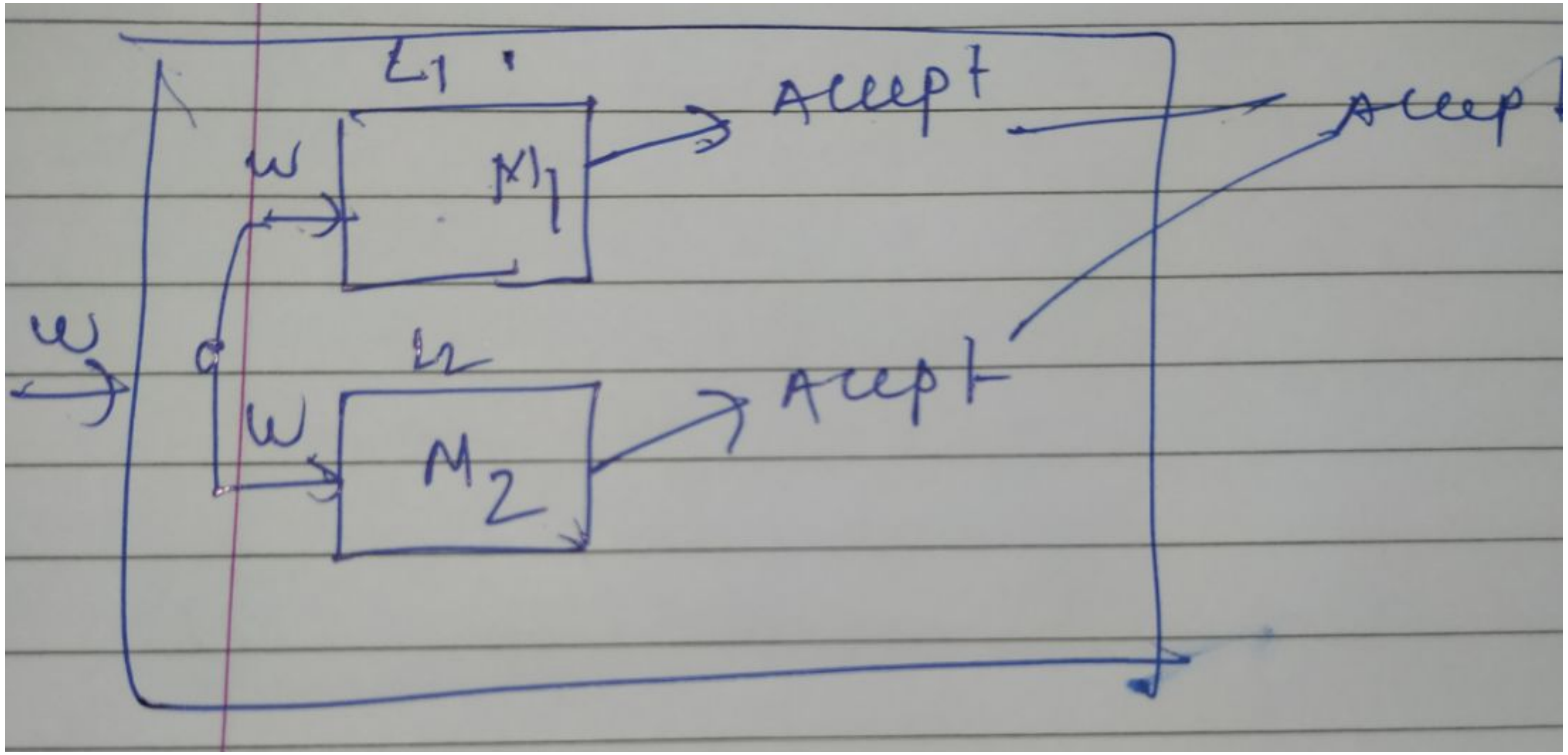


Figure 9.4: Simulation of two TM's accepting a language and its complement

If input w to M is in L, then M_1 will eventually accept. If so, M accepts and halts. If w is not in L, then it is in $\overline{L}$, so M_2 will eventually accept. When M_2 accepts, M halts without accepting. Thus, on all inputs, M halts, and $L(M)$ is exactly L. Since M always halts, and $L(M) = L$, we conclude that L is recursive. □

Union of 2 Recursively Enumerable languages is also Recursively Enumerable



Let M1 be a TM accepting language L1 and M2 be a TM accepting L2.
M be TM accepting L1 union L2.
M accepts if either M1 or M2 accepts
M rejects if either M1 and M2 rejects.

Tractable & Intractable Problems

constant	$O(1)$
logarithmic	$O(\log n)$
linear	$O(n)$
n-log-n	$O(n \times \log n)$
quadratic	$O(n^2)$
cubic	$O(n^3)$
exponential	$O(k^n)$, e.g. $O(2^n)$
factorial	$O(n!)$
super-exponential	e.g. $O(n^n)$

Computer Scientist divides these functions into 2 classes

Polynomial functions: Any function that is $O(n^k)$, i.e. bounded from above by n^k for some constant k.
E.g. $O(1)$, $O(\log n)$, $O(n)$, $O(n \times \log n)$, $O(n^2)$, $O(n^3)$

Exponential functions: The remaining functions.
E.g. $O(2^n)$, $O(n!)$, $O(n^n)$

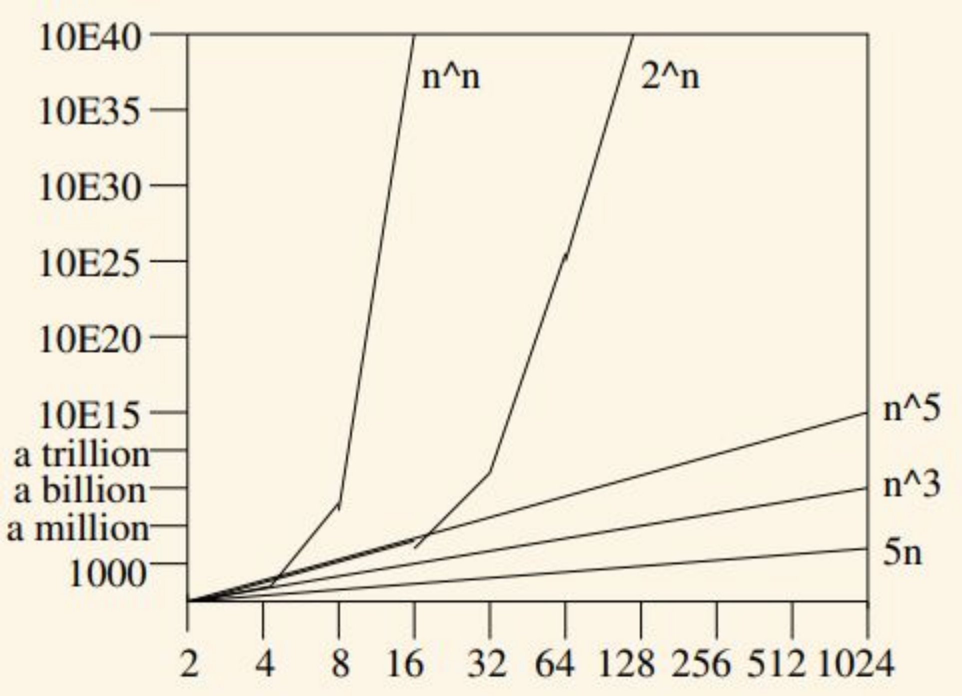
$a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0.$

But here the word 'polynomial' is used to lump together functions that are bounded from above by polynomials. So, $\log n$ and $n \times \log n$, which are not polynomials in our original sense, are polynomials by our alternative definition, because they are bounded from above by, e.g., n and n^2 respectively.

Tractable & Intractable Problems

Polynomial-Time Algorithm: an algorithm whose order-of-magnitude time performance is bounded from above by a polynomial function of n, where n is the size of its inputs.

Exponential Algorithm: an algorithm whose order-of-magnitude time performance is not bounded from above by a polynomial function of n.



Tractable & Intractable Problems

In the similar way we classify problem into 2 classes

Tractable Problem: a problem that is solvable by a polynomial-time algorithm. The upper bound is polynomial.

Intractable Problem: a problem that cannot be solved by a polynomial-time algorithm. The lower bound is exponential.

Here are examples of tractable problems (ones with known polynomial-time algorithms):

- Searching an unordered list
- Searching an ordered list
- Sorting a list
- Multiplication of integers (even though there's a gap)
- Finding a minimum spanning tree in a graph (even though there's a gap)

- Here are examples of intractable problems (ones that have been proven to have no polynomial-time algorithm).
  - Some of them require a non-polynomial amount of output, so they clearly will take a non-polynomial amount of time, e.g.:
    - Towers of Hanoi: we can prove that any algorithm that solves this problem must have a worst-case running time that is at least $2^n - 1$.
    - List all permutations (all possible orderings) of n numbers.

The Halting Problem

Given a Program, WILL IT HALT ?

Given a Turing Machine, will it halt when run on some particular given input string?

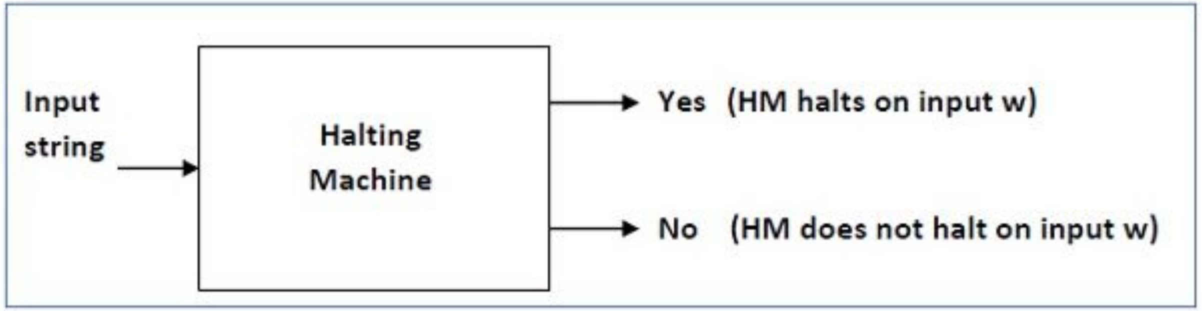
Given some program written in some language (Java/C/ etc.) will it ever get into an infinite loop or will it always terminate?

Answer:

- In General we can't always know.
- The best we can do is run the program and see whether it halts.
- For many programs we can see that it will always halt or sometimes loop

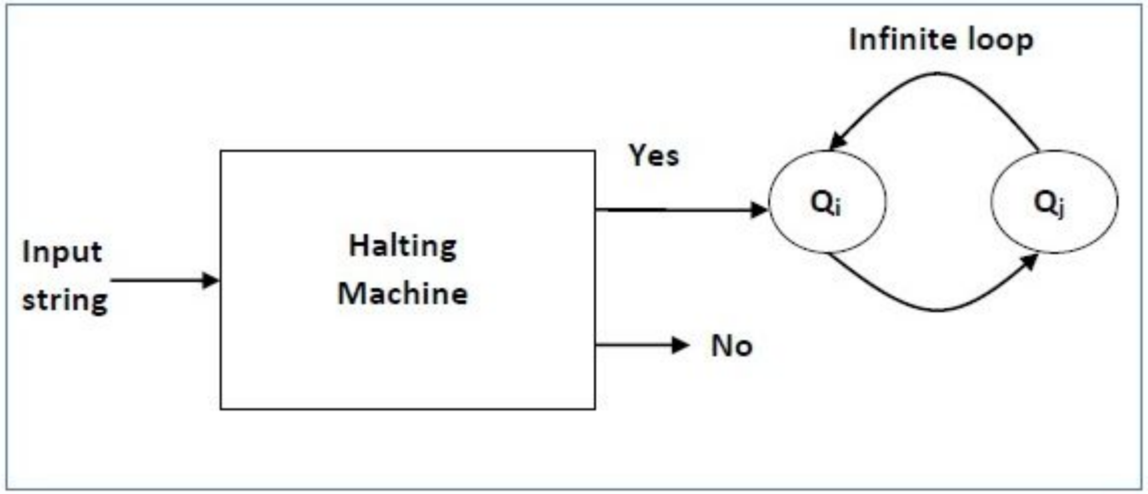
Proof of Halting problem of TM is undecidable

Proof − At first, we will assume that such a Turing machine exists to solve this problem and then we will show it is contradicting itself. We will call this Turing machine as a **Halting machine** that produces a 'yes' or 'no' in a finite amount of time. If the halting machine finishes in a finite amount of time, the output comes as 'yes', otherwise as 'no'. The following is the block diagram of a Halting machine −



Now we will design an **inverted halting machine (HM)'** as −

- If **H** returns YES, then loop forever.
- If **H** returns NO, then halt.



a machine **$(HM)_2$** which input itself is constructed as follows ·

If $(HM)_2$ halts on input, loop forever.

Else, halt.

This is a Contradiction hence halting problem of TM is undecidable.