TM – Undecidability

Course Instructor: Jibi Abraham

Decision Problems

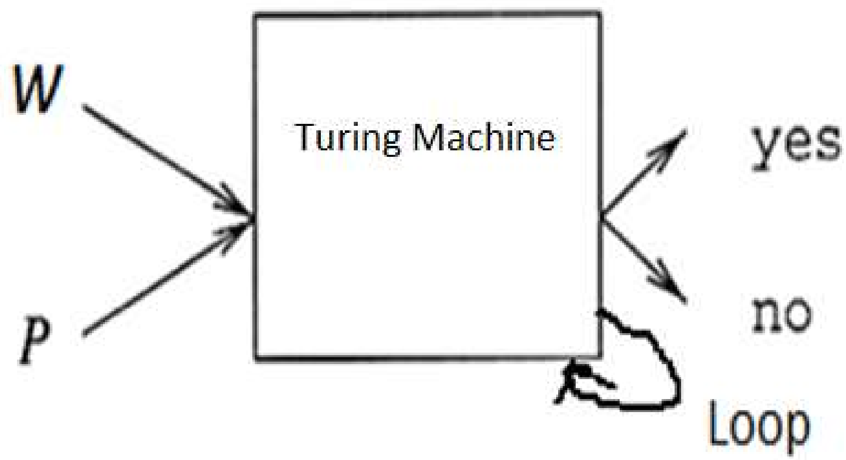
- A decision problem is a problem for which the answer is “yes” or “no”
- For example, a CFG is ambiguous or not is a decision problem
- Decision problems are divided into two categories

1. There are some problems which TMs can solve (TM Recognizable)
2. There is no TM possible to solve the problem (Non-Recursively Enumerable)

TM Recognizable Problems

- A language L is Recursive Enumerable (RE), if L = L(M) for some TM M
- Also known as TM Recognizable

TM Recognizable



- A decision problem P is TM Recognizable if there is a TM which
  - if the TM accepts the input *w:* will always halt in a finite amount of time to give an answer as 'yes'
  - if the TM rejects the input *w:* the answer will be either “no”, and the TM may halt or will be in a loop

TM Recognizable - Example

- Halting program: A program with a simple print statement

  print("Hello World”);

- Non-halting program: An infinite loop like

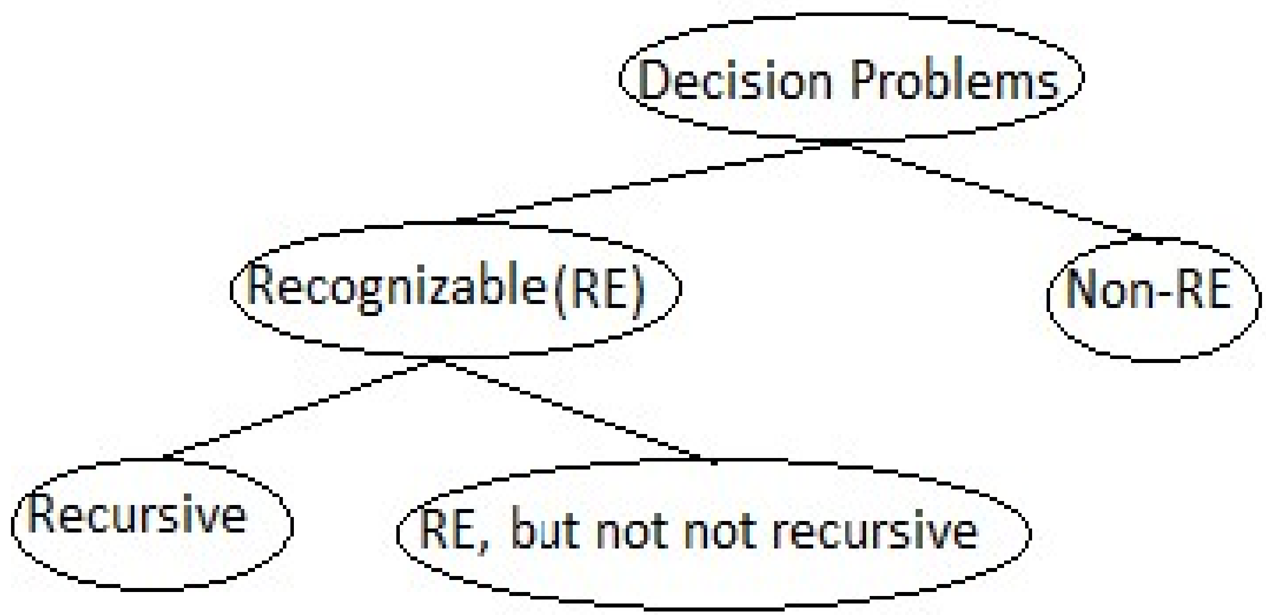
  while (true) {}

```cpp
#include <iostream>
using namespace std;

int main() {
  // A halting code
  cout << "Hello World"<<endl;
  // An infinite loop
  while(true){
    cout<<""<<endl;
  }
  return 0;
}
```

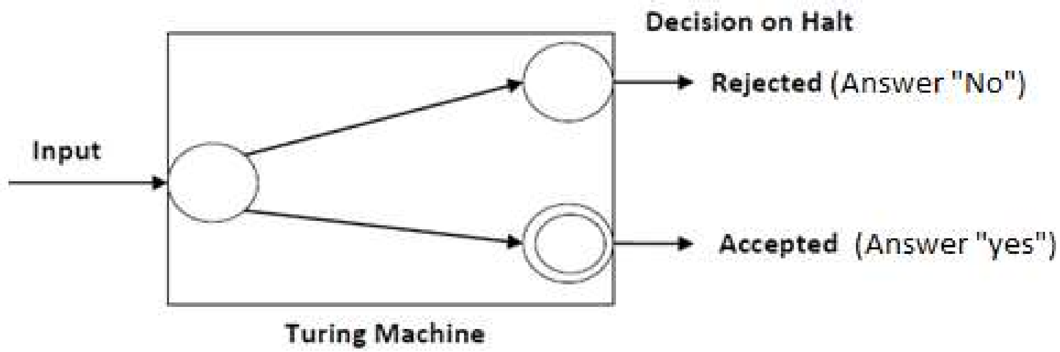
TM Recognizable Problems

- The problems which can be solved by a TM (TM Recognizable Problems) can be categorized as
  - Decidable (Recursive Languages)
  - Undecidable



Decidable Problems

- A decision problem P is decidable if there is a TM with algorithm H which will always halt in finite amount of time to give answer as 'yes' or 'no' for a given input *w*



- The answer will be “yes” if the TM accepts the input w and the answer will be “no” if the TM rejects the input w
- After accept or reject, the TM will surely halt, regardless of whether it accepts or not
- This TM is a good model of an “algorithm”
- The language accepted by a Decider TM is called recursive language

Church's Thesis

- Church-Turing Thesis formalized the notion of an algorithm, as a procedure that can be performed by a decider TM
- Church's Thesis states that all sufficiently powerful and reasonable models of computation belong to the same class
- By accepting Church's Thesis, we are able to prove that certain problems are unsolvable (undecidable) by any computer
- ie. Some decision problems do not have an algorithm that can be performed by a TM

Decision Problem – Example 1

- Is the problem to check a number ‘m’ prime decidable?
- Algorithm:
- Divide the input number **‘m’** by all the numbers between ‘2’ and ‘$\sqrt{m}$’ starting from ‘2’. If any of these numbers produce a remainder zero, then it goes to the “Reject” state, otherwise, it goes to the “Accept” state. So, here the answer could be made by ‘Yes’ or ‘No’.
- Hence, it is a decidable problem

Recursive Languages - Example

- Theorem 1: Every regular language L defined by a DFA is decidable
- Proof:
- Let D be a DFA with L(D) is a regular language. Design a TM T that simulates D.
- After processing the input, if the simulated D is in an accepting state, T accepts; else T rejects.
- Algorithm: T = On input <D, w>,
  1. Simulate D on input w
  2. If the simulation ends in an accept state, *accept* – otherwise, *reject*.”
- TM would keep track of the current state and process the input tape based on transition function and finally accept if the string’s final state is accept
- Hence DFA defining regular language is decidable

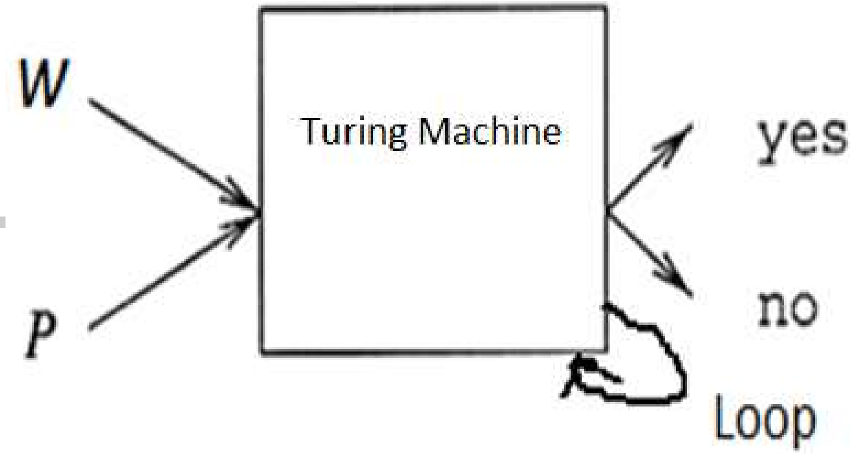
Recursive Languages - Example

- Theorem 2: The language defined by NFA is decidable
- **Proof:**
- We construct a TM T, deciding the language of NFA N
- Algorithm: T = On input <N, w>,
- 1. Convert N to a DFA D.
- 2. Run T from proof in Theorem 1, on <D, w>
- 3. If D reaches a final state, accept w, else reject w

Other Recursive Languages

- Language of Emptiness (accepts empty string) of Finite Automata
  - Algorithm: Check if a final state is reachable from the start state
- Language defined by Regular Expression
  - Algorithm: Convert the regular expression to an equivalent NFA
- Two DFAs recognizing the same language
- Language defined by Context Free Language
- Language of Emptiness of CFG
- Two CFGs generating the same language

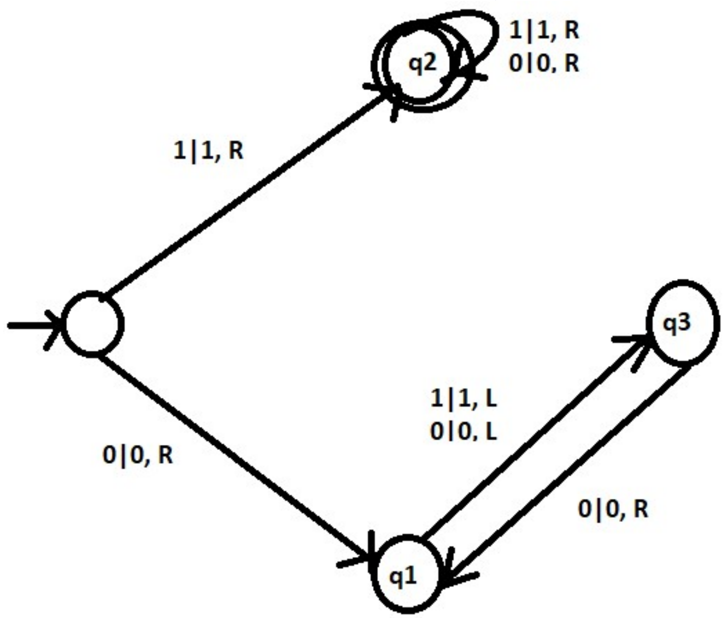
Undecider TM



- This type of TMs fail to halt on some input strings
- Example 1: All polynomial equations with integer coefficients that have a solution in the integers
  - TM accepts $x^3+y^3+z^3 = 0$
  - Rejects $x^2+y^2+1 = 0$
  - But loops on the input $x^4+2y^3+z^4 = 5$

Undecider TM - Example

- Example 2: TM recognizing Regular Exp 1(0+1)*



  - TM accepts strings which starts with a 1 and halts
  - TM rejects string starts with a 0, but loops without halting
- Hence, the language is RE, but non-Recursive
- There cannot be any algorithm to recognize them

Undeciadable Languages- Example

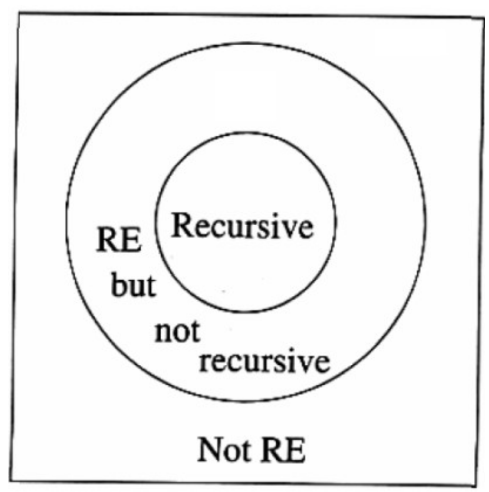
- Given a CFL, there is no TM which will always halt in finite amount of time and give answer whether language is ambiguous or not
- Given two CFL, there is no TM which will always halt in finite amount of time and give answer whether two CFLs are equal or not
- Given a CFG and input alphabet, whether CFG will generate all possible strings of input alphabet ($\sum^*$) is undecidable
- Given a CFL, CSL or RE, determining whether this language is regular is undecidable

Popular Undeciadable Problems

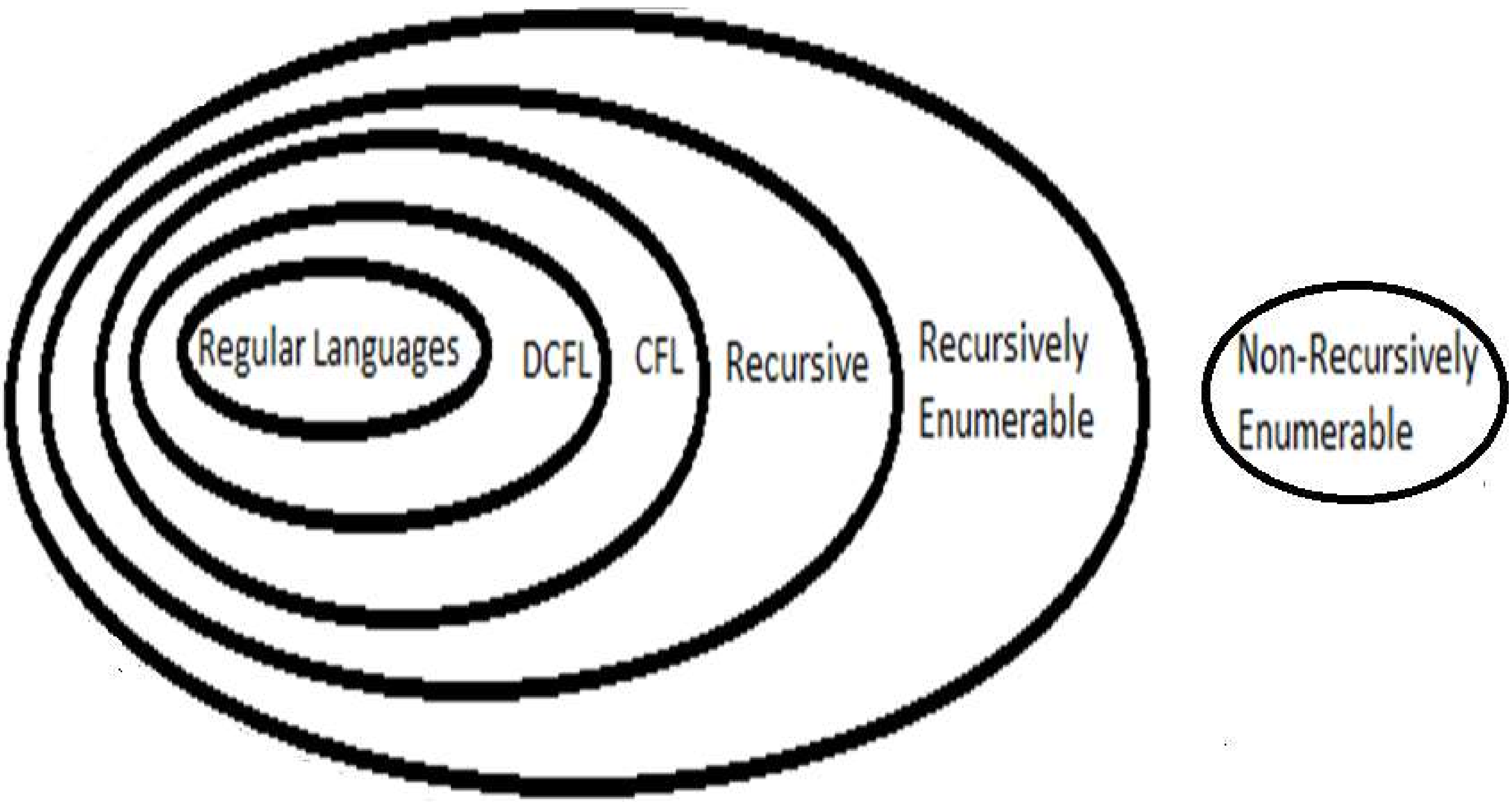
- T*wo* popular *Undecidable problems* are:
  - Halting problem of TM
  - Post Correspondence Problem (PCP)

Non-RE Language

- A language is Non-Recursively Enumerable, if there is no Turing Machine that accepts the language.
- Relationship between the languages of decision problems (Recursive, Recursively Enumerable, but not recursive and Non-Recursively Enumerable) is shown as



Relationship among Language Classes



Encoding of TM

- In many proofs involving TMs, we need to enumerate the binary strings and encode TMs so that we can refer to the i^{th} binary string as w_i and the i^{th} TM as M_i
- Binary strings are easy to enumerate
- If w is a binary string, shall treat w number of 1's as the binary integer i
- Eg: the empty string is the first string, 0 the second, 1 the third, 00 the fourth, 01 the fifth, and so on
- Hence forth, will refer to the i^{th} string as w_i

Encoding for TM

- A binary code for all TMs with the input alphabet {0, 1} so that each TM can be represented by a binary string
- States of TM are $q_1, q_2, \ldots, q_r$ for some r with q_1 the start and q_2 the only accepting state
- Tape symbols are $X_1, X_2, \ldots, X_s$ for some s with X_1 as 0, X_2 as 1 and X_3 as the blank
- Integers D_1 and D_2 as tape head directions left and right
- Encode a transition rule $\delta(q_i, X_j) = (q_k, X_l, D_m)$ as a binary string C of the form $0^i10^j10^k10^l10^m$
  - 1 acts as a delimiter;
  - first 0^i represents the state q_i, the next 0^j represents the tape symbol X_j, the next 0^k, represents the transiting to state q_k, the next 0^l represents the tape symbol X_l which replaces the tape symbol X_j and the last 0^m represents the direction of tape head move D_m

Encoding for TM

- Suppose there are n transition rules
- Binary code for the entire TM will be the concatenation of the codes for all of the transitions separated by pairs of 1's:
  - $C_1 11 C_2 11 \ldots C_{n-1} 11 C_n$
- There can be many encodings for the same TM
- An encoding pair (M_i, w) consisting of encoding of the TM M_i, then seperator,111 and then the string w
- Thus, a binary code can be assigned for every TM

Example

- What will be the encoding of the following TM:

	0	1	B
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	(q_2, B, L)
q_2	$(q_3, 0, R)$	-	-
q_3	-	-	-

q_1,	0
q_2,	00
q_3	000
:	
0	0
1	00
B	000
L	0
R	00

Example:

$\delta(q_2, 1) = (q_3, 0, R)$

Is encoded as:

00100100010100

encoding of a TM

*111*010101010011010010100100110100010010001011001010001010011*111*

Diagonalization Method

- For finite sets, we can simply count the elements
- But it is not possible to count the number of elements in an infinite set
- Do infinite sets N= {0, 1, 2, … } and Z= {1, 2, 3, … } have the same size?
- N is larger because it contains an extra element 0 and all other elements of Z

Diagonalization Method (contd)

- Technique of diagonalization was discovered in 1873 by Georg Cantor
- He was concerned with the problem of measuring the sizes of infinite sets
- He observed that two finite sets have same size if elements of one set can be paired with the elements of the other set
- This pairing can also be extended to infinite set through a mapping/ correspondence/ bijection function (one-to-one (injection) and onto (surjection)) between them

Diagonalization Method (contd)

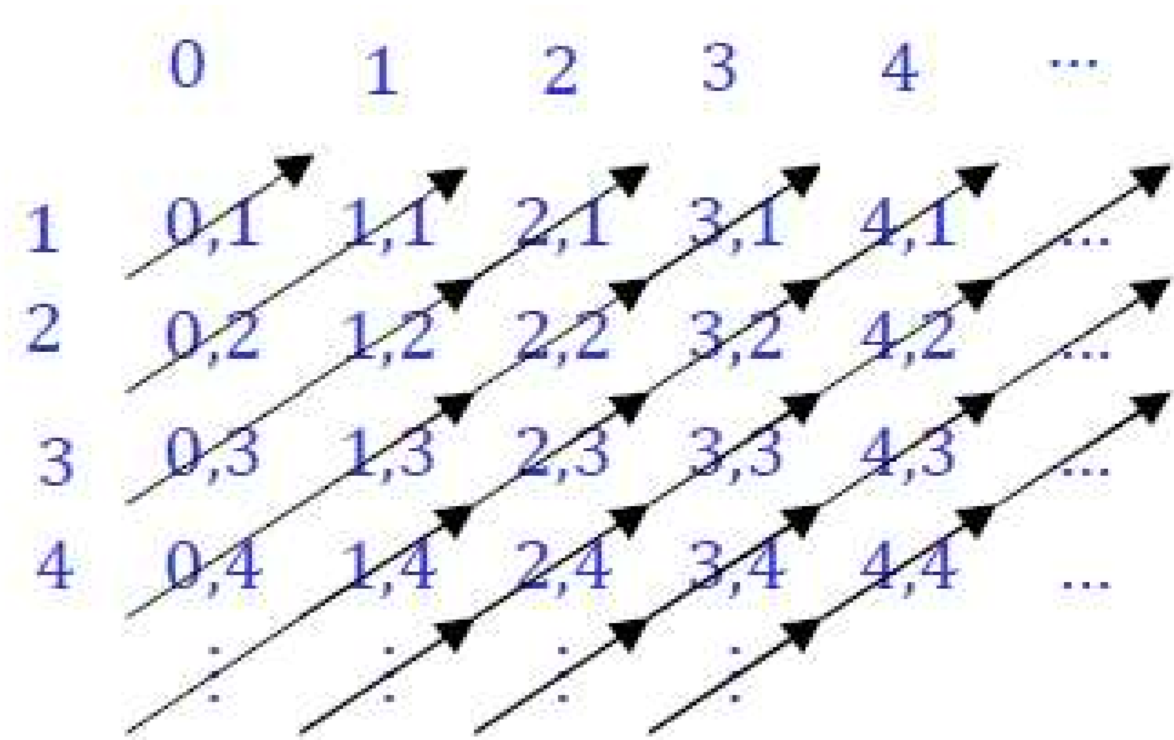
- Bijection function satisfies:
  - One-to-one means a ≠ b implies f(a) ≠ f(b)
  - Onto means for every element b in 2nd set, there is some a in the first set such that f(a) = b
- For example, the mapping f(k) = 2k is a bijection between the integers and the even integers

Countable Infinite Set

- A set is said to be ***countable***, if there exists a bijection between it and the set of naturals N = {0, 1, 2, 3, 4, …}
- ie. an infinite set that has the same size as N is countable
- To show that any infinite set Z is countable, establish a mapping with the set of natural numbers, N
- We pair the 1^{st} element of Z with 1^{st} element of N, 2^{nd} element of Z with 2^{nd} element of N and result to an infinite pairing matrix

Countable Set Example

- For example, The pairing created between the elements of N = {0, 1, 2, ... } and Z= {1, 2, 3, ... } results in the in a Infinite Pairing Matrix with mapping



Bijection between N and Z

- Infinite sets N = {0, 1, 2, ... } and Z= {1, 2, 3, ... } have the same size because each element of Z can be mapped to a unique element of N by using the bijection function *f* between Z and N: $f(z) = z-1$

$$|N| = |Z| \quad \begin{array}{ccccc} 0 & 1 & 2 & 3 & \ldots \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow & \\ 1 & 2 & 3 & 4 & \ldots \end{array}$$

Infinite Pairing Matrix for N to Z

- To get the infinite pairing matrix for N to Z mapping
  - Pairing is created with i^{th} row containing all numbers from Z and j^{th} column containing all numbers from N
  - The i^{th} row, j^{th} column element is 1 if the bijection is satisfied, else is 0
  - Bijection between Z and N: $f(z) = z-1$ and the Infinite pairing matrix thus created is

$$
\begin{array}{cc|ccccccc}
 & & & & & N & & & \\
 & & 0 & 1 & 2 & 3 & 4 & 5 & \cdots \\
\hline
 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots \\
 & 2 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots \\
Z & 3 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots \\
 & 4 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots \\
 & 5 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots \\
 & 6 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots \\
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\
\end{array}
$$

Countable sets -Example

- Example 2: Refer to Sipser textbook example to show that set of positive rational numbers is countable
- Example 3: The set of all binary sequences of finite length is countable
  - Let ϵ denote the empty sequence (with no terms)
  - Then, the infinite sequence is ϵ, 0, 1, 00, 01, 10, 11, 000, 001,...
  - This set contains binary sequences of length 0, then the sequences of length 1 listed in increasing numeric order, then the sequences of length 2 are listed in increasing numeric order and so on
  - Set contains every binary sequence of finite length exactly once. Hence the set is countable

Exercise

- Prove that the set of all integers that are multiples of 5 is countable

Uncountable Infinite Set

$$
\begin{array}{c|cccccccc}
 & \multicolumn{7}{c}{N} \\
 & 0 & 1 & 2 & 3 & 4 & 5 & \cdots \\
\hline
1 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots \\
2 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots \\
Z\ 3 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots \\
4 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots \\
5 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots \\
6 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots &
\end{array}
$$

- Consider the diagonal of any pairing matrix $d(f)=(f(0), f(1), f(2),...)$
- Ex: in the Pairing Matrix between N= {0, 1, 2, ...} and Z= {1, 2, 3, ... } $d(f) = (1, 1, 1, 1, \ldots)$
- Complement of the diagonal, $d^c(f)$ is $(0, 0, 0, \ldots)$
- It can be seen that $d^c(f)$ do not match with any matrix row
- If $d^c(f)$ is considered as a row of the Matrix, it disagrees in some columns with the Bijection function, which contradicts the paring between the sets

Uncountable Set (contd)

- Such infinite sets, where no bijection with N exists is said to be uncountable
- Uncountable infinite set theory is very useful to prove that some languages are unrecognizable
- Ex: Set of real numbers
  - To prove that some infinite set is uncountable, Cantor used the Diagonalization method

Uncountable -Example

N	Real Number
1	0.~~2~~3246…
2	0.3~~0~~589…
3	0.21~~7~~54…
4	0.054~~2~~4…
5	0.9954~~8~~…
.	….
.	….

- Theorem: Set of real Numbers R is uncountable
- Proof:
- In order to show that R is uncountable, show that no a bijection exists between N and R
- The proof is by contradiction
- Assume that a bijection *f* exists between N and R
- Then *f* must pair all members of N with all the members of R, *f*(1) = 0.23246, *f*(2) = 0.30589 etc
- But, we will find an x in R that is not paired with any other number in N, which will be a contradiction

Theorem Proof

N	Real Number
1	0.~~2~~3246…
2	0.3~~0~~589…
3	0.21~~7~~54…
4	0.054~~2~~4…
5	0.9954~~8~~…
.	….
.	….

- We find such an x between 0 and 1 by constructing it
- We choose each digit d_j of x as:
  - d_j= 0 if digit j of j^{th} Real number is greater than 0
  - d_j = 1 if digit j of j^{th} Real number is equal to 0
- Get a unique x as 0.01000…, which is not equal to any real number in the set. ie. $x \neq f(n)$ for any $n \in N$
- So there a pair which do have not a bisection, which contradicts the assumption
- Hence, proved that the set of real numbers is uncountable

Exercise

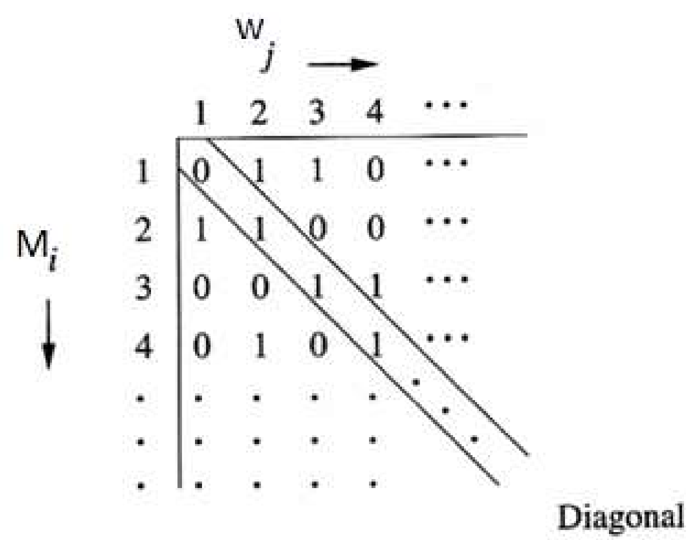
- Prove that the set of all binary sequences of infinite length is uncountable

Diagonalization Language

- Diagonalization as a proof technique is used to demonstrate that there are some languages that cannot be recognized by a TM
- L_d, the diagonalization language is defined:
  - Let $w_1, w_2, w_3, \ldots$ be an enumeration of all binary strings
  - Let $M_1, M_2, M_3, \ldots$ be an enumeration of all TMs
  - Let $L_d = \{w_i \mid w_i \text{ is not in } L(M_i)\}$.
- L_d consists of all strings of w_i such that the TM M whose code is M_i, does not accept when given w_i as input

Pairing Matrix for L_d

- Create a pairing matrix consists of each row corresponds to a TM M_i and each column w_j corresponds to the input string
- (i, j) entry of the matrix is 1 if M_i accepts w_j and 0, otherwise
- i^{th} row is the characteristic vector for the language $L(M_i)$; the 1's in the row indicates all the strings of the language accepted by M_i



Pairing Matrix for L_d

$w_j \longrightarrow$

$M_i \downarrow$

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & \cdots \\ \hline 1 & 0 & 1 & 1 & 0 & \cdots \\ 2 & 1 & 1 & 0 & 0 & \cdots \\ 3 & 0 & 0 & 1 & 1 & \cdots \\ 4 & 0 & 1 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

- Diagonal values of this matrix tell whether M_i accepts w_i
- To construct L_d (all w not in L(M)), we complement the diagonal
- In this example, the complement of the diagonal is (1, 0, 0, 0, …) meaning that w_1 is not in M_1, w_2, w_3 and w_4 are in M_2, M_3, M_4 etc
- It can be seen that the diagonal cannot be the characteristic vector of any Turing Machine in any row
- This trick of complementing the diagonal is called diagonalization

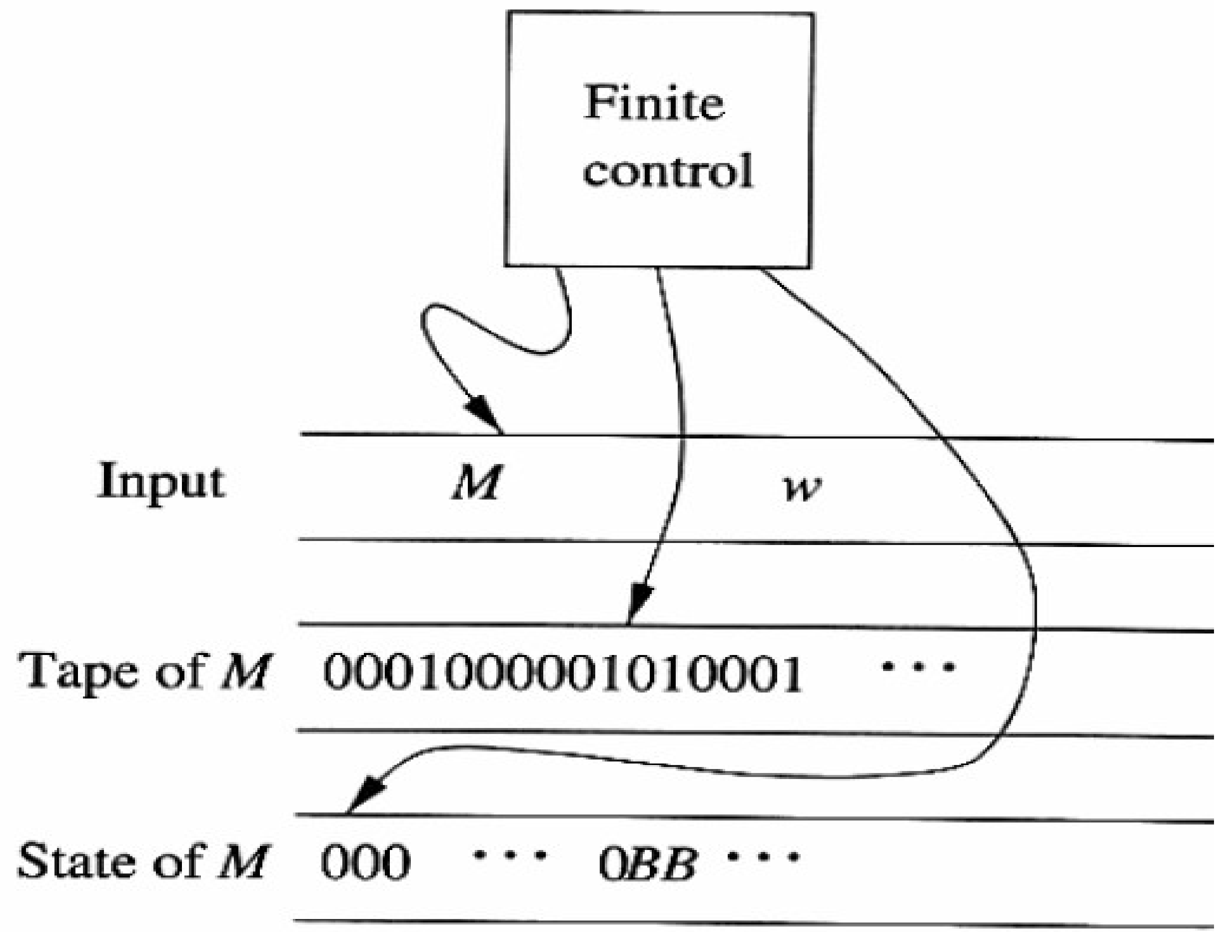
L_d is non-RE

- Theorem: L_d is not a RE language
- Proof: We have to prove that there is no TM to accept L_d.
- Suppose $L_d = L(M_i)$ for some TM M_i
  - This gives rise to a contradiction. Consider what M_i will do on an input string w_i.
    - If M_i accepts w_i, then by definition w_i cannot be in L_d.
    - If M_i does not accepts w_i, then by definition w_i is in L_d.
  - Since w_i can neither be in L_d nor not be in L_d, we must conclude there is no TM that can define L_d.
- L_d does not have any TM to accept it. Hence, L_d is not recursively Enumerable

Universal Turing Machine

- A Universal Turing Machine is a fixed Turing Machine which can simulate any Turing Machine M including itself for the purpose of language recognition
- UTM has three tapes:
- 1st tape contains the encoding of the TM M
- 2nd tape contains $w = a_1a_2...a_k \in \{0, 1\}^*$ in the form $00010^{a1+1}10^{a2+1}1 \ldots 10^{ak+1}10000$
  - It simulates the tape contents of M, with the tape head keeps changing the tape contents as the simulation proceeds and the tape head keeps moving left or right
- 3rd tape keeps track of the state of the TM M during the simulation

Organization of UTM



Working of UTM

- For a TM M = (Q, X, ∑, δ, q_0, B, F) to validate a string w, the UTM takes the binary code as <M, w> and simulates M on w
- UTM works in phases, simulating one transition of M at each step
- First, UTM searches the position of the encoding of M (tape 1) that corresponds to the simulated state (tape 3) of M and the symbol in tape 2 at the position of the tape head 2
- Let the chosen sequence of encoding be $0^{i+1}1^{0j+1}10^{r+1}10^{s+1}10^{t+1}$, which corresponds transition function rule $\delta(q_i, a_j) = (q_r, a_s, \Delta_t)$.
- During the simulation, state 0^{i+1} in tape 3 will be changed to state 0^{r+1} and on tape 2, the tape symbol 0^{j+1} will be replaced to 0^{s+1}
- In addition, the head of tape 2 will be moved to the left so that the code of one symbol is passed, if t = 0 and to the right otherwise

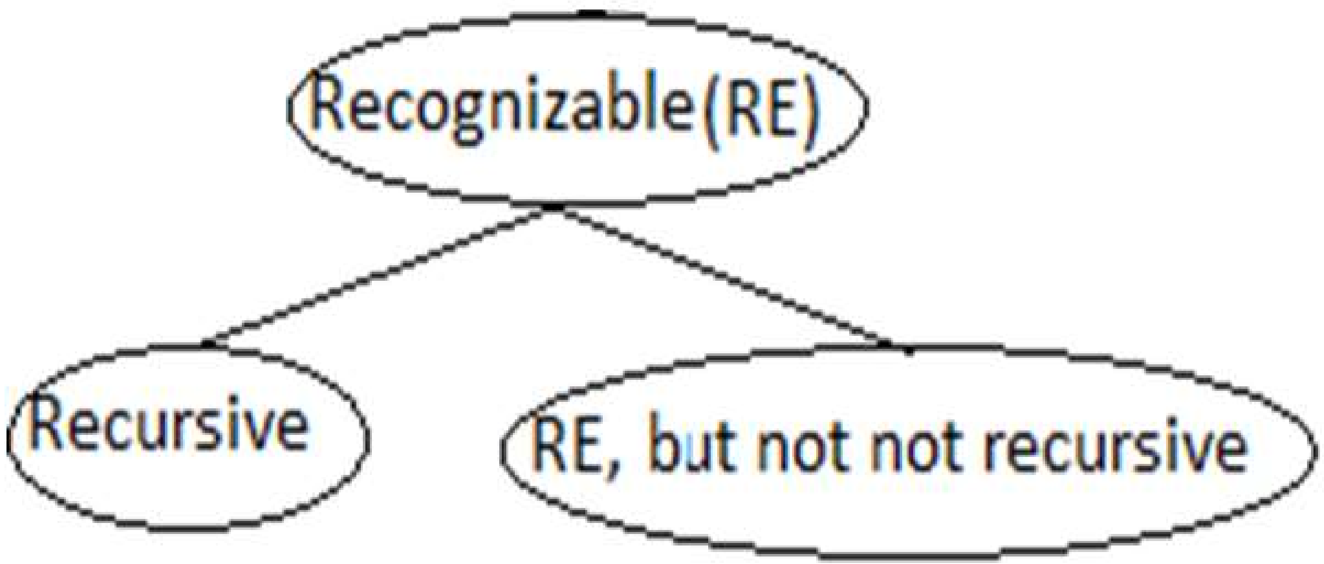
Working of UTM

- When tape 1 does not contain any code for the simulated state q_i, M has reached a final state to **accept** or **reject**
- TM M may not halt when the input string w is not in the language, UTM may also not halt, thus will have the same behaviour as M on w
- UTM accepts <M, w> if and only if M accepts w

Universal Language

- The Universal language L_u is the set of binary strings that encode a pair <M, w> (by putting111 between the code for M and w) where $w \in L(M)$ so that $L_u = L(M)$
- L_u is RE, but, not recursive (Proof given in Hopcroft text book)

Closure Properties of TM Languages



Closure Properties

Property	Recursive Languages	Recursively Enumerable, non Recursive Languages
Union	Closed	Closed
Intersection	Closed	Closed
Concatenation	Closed	Closed
Kleene closure	Closed	Closed
Complementation	Closed	Not Closed

Theorem 1

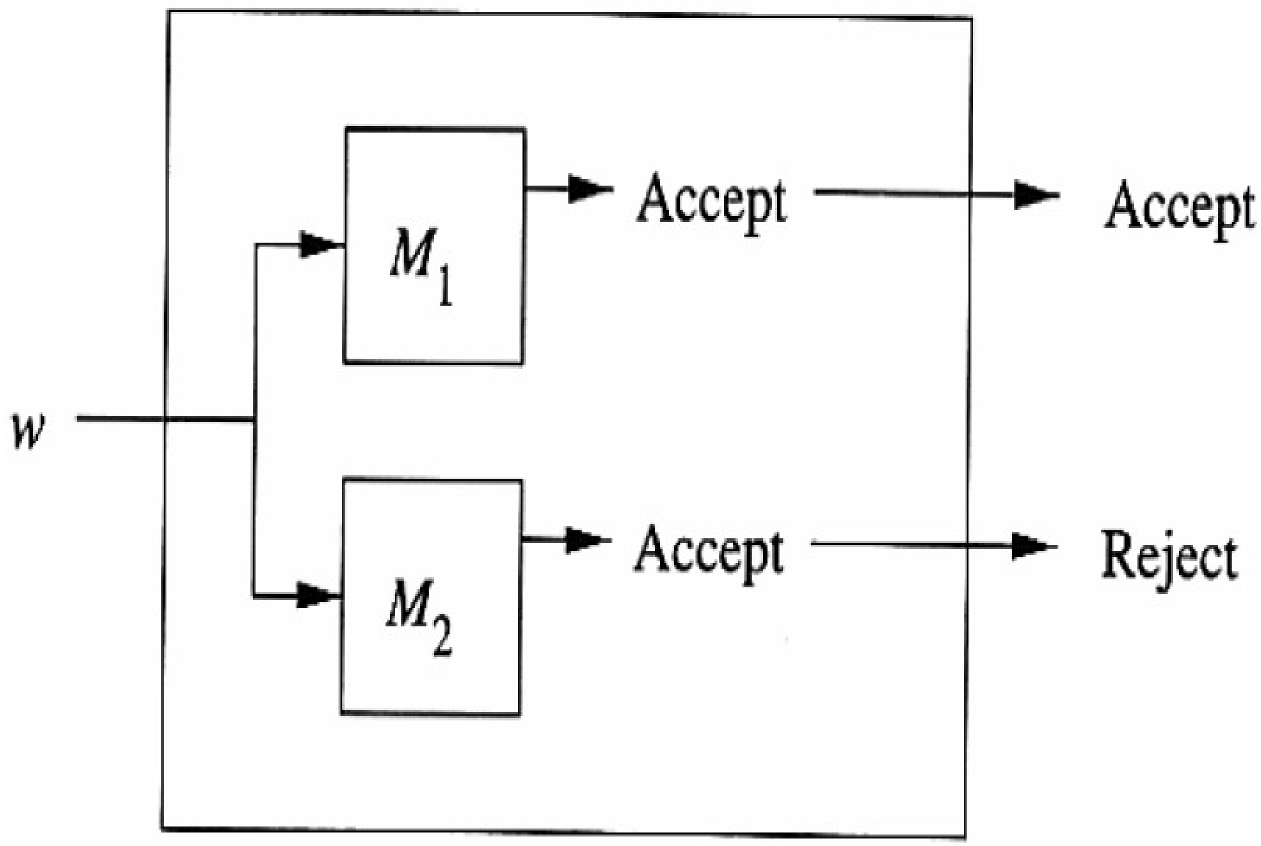
- If L is recursive, then L^c is recursive
- Proof:
- Let L and L^c be recognizable by M_1 and M_2, respectively. We construct machine M that decides L:
- M = “On input w,
- Set n = 1
  1. Simulate M_1 on w for n steps. If it accepts, accept
  2. Simulate M_2 on w for n steps. If it accepts, reject
  3. Increment n and go to step 1”
- Either $w \in L$, or $w \notin L$. Therefore either M_1 or M_2 will halt in a finite number of steps. Therefore, M will halt in a finite number of steps. Hence L is recursive.

Theorem 2

- If L and its complement L^c are RE, then L is recursive
- Proof:
- Every recursive language is also RE
- Let $L = L(M_1)$ and $L^c = L(M_2)$
- Construct a TM M that simulates M_1 and M_2 in parallel (using two tapes and two heads)
- If the input to M is in L, then M_1 accepts it and halts, hence M also accepts it and halts
- If the input to M is not in L, then M_2 accepts it and halts, hence M halts without accepting it
- Clearly, any string belongs to either L or L^c
- Hence, any string will cause either M_1 or M_2 (or both) to halt
- Hence, M halts on every input and $L(M) = L$, so L is recursive

Theorem 2 (Contd)

- TM M which simulates L and L^c is given as



Halting Problem of TM

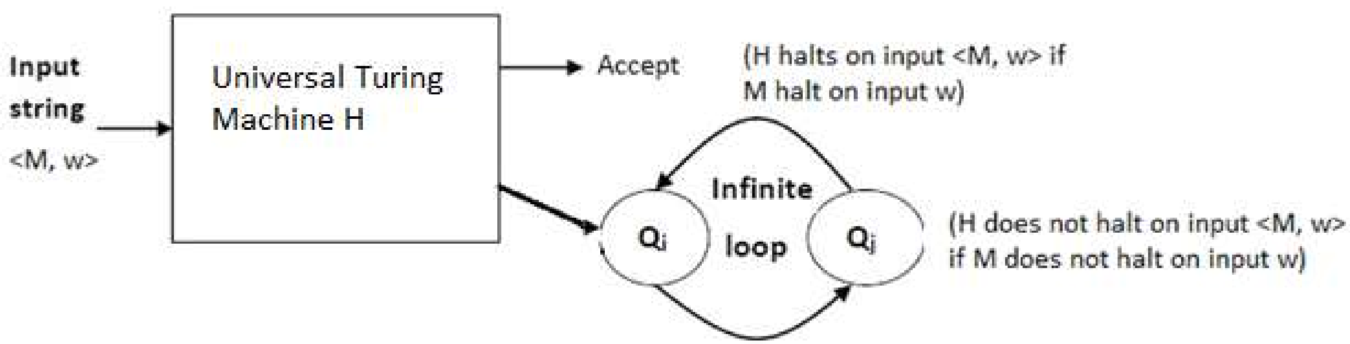
- For an undecidable language, there is no TM which accepts the language and makes a decision for every input string w (TM can make decision for some input string though)
- Alan Turing proved the existence of undecidable problems in 1936 by finding an example, the now famous "halting problem“
- Definition: “*Based on its code and an input, will a particular program ever finish running?*”

Halting Problem of TM

- Is there a TM which has two parameters: A TM M and an input w and which returns: YES when M stops on w and NO when M loops on w?
- If there is such a TM, then Halting problem is undecidable; otherwise, it is decidable
- Result from [Turing 1936] shows that Halting problem is undecidable
- A halting problem is the problem of determining whether a TM finish running on an input string in a finite number of steps

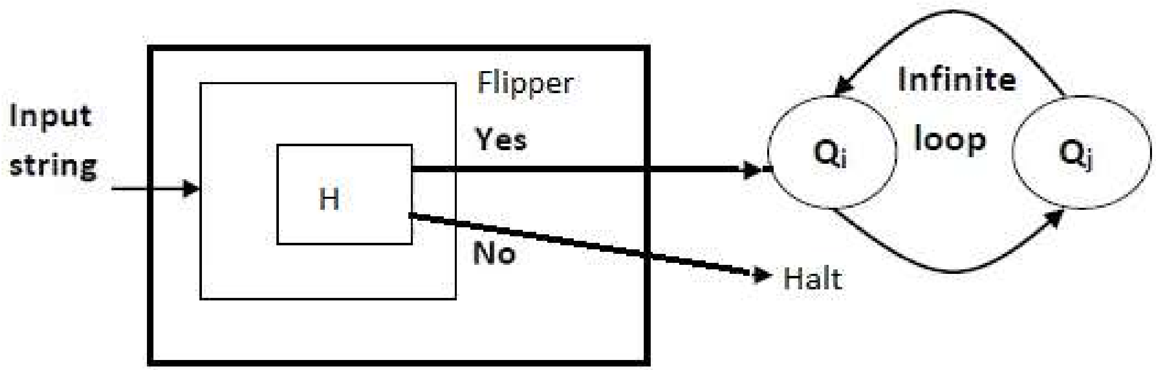
Halting Problem Theorem

- $A_{TM} = \{\langle M, w\rangle | M$ is a Turing machine that accepts $w\}$ is not TM decidable.
- Proof: assume that A_{TM} is decidable, in order to obtain a contradiction
- If A_{TM} is decidable, there is some Universal TM, H decides it
- H will accept the input ($\langle M$, w⟩), if M accepts w and reject if M does not accept w (either by rejecting or looping indefinitely)



Halting Problem Proof

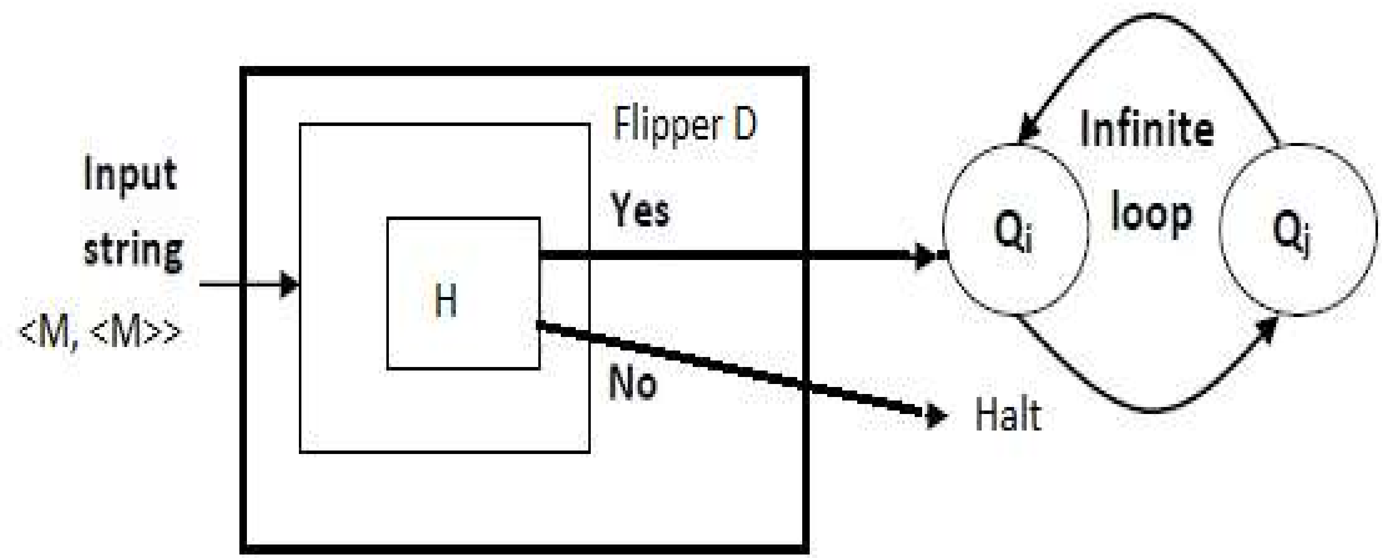
- Build a new TM Flipper which calls UTM H
- Flipper rejects the input if H accepts and Flipper accepts, if H rejects



- Using the nature of Flipper, we build a TM D as
  - D="On input $\langle M\rangle$, where M is a Turing machine:
    - Run H on input $\langle M,\langle M\rangle\rangle$
    - Do the opposite of H. If H accepts, then reject. If H rejects, than accept."

Halting Problem Proof

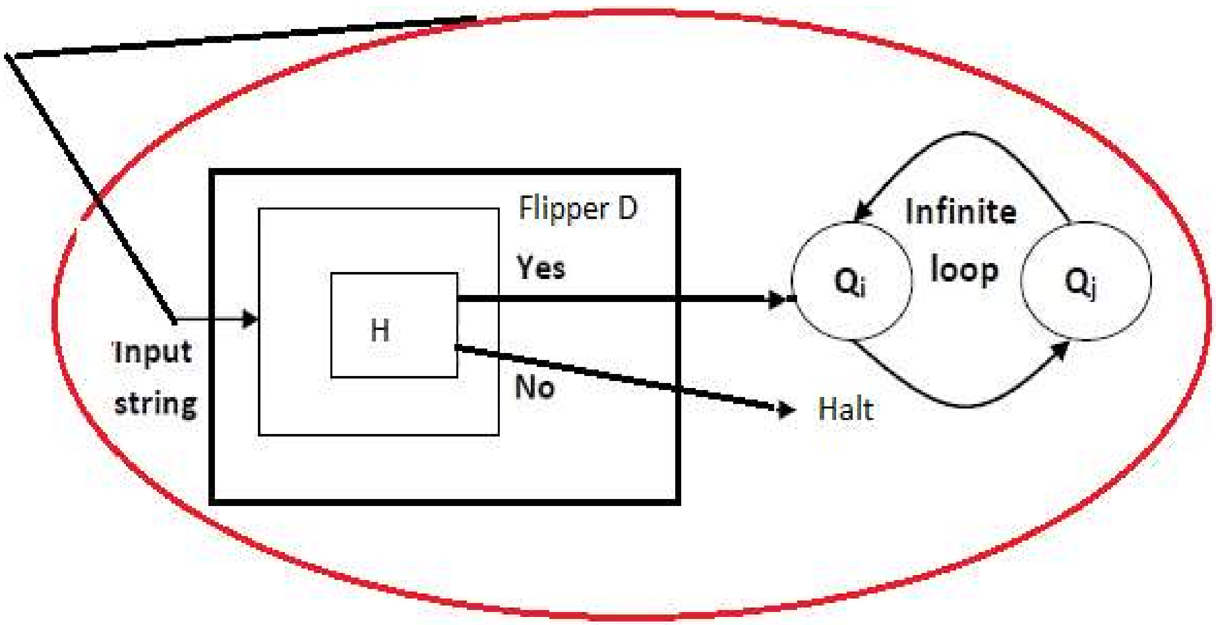
- Let Flipper TM D takes a TM M as input, then runs the TM H to see if M accepts its own description
- If M accepts itself, then D rejects and vice versa
- Notice that since H always halts with either acceptance or rejection, D always halts



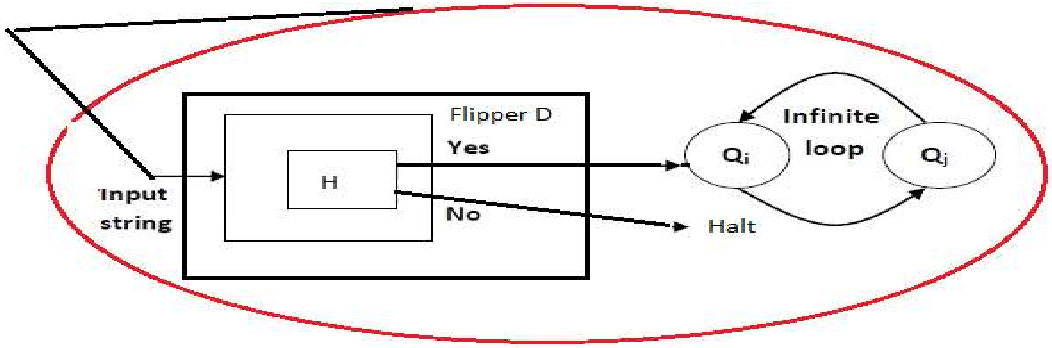
Halting Problem Proof

- If we run the flipper D on itself (Flipper on Flipper as input), then D will run itself on its own description (<D, <D>>)

$$\text{Flipper}\left(\langle\text{Flipper}\rangle\right) = \begin{cases} \text{reject} & \text{Flipper accepts } \langle\text{Flipper}\rangle \\ \text{accept} & \text{Flipper does not accept } \langle\text{Flipper}\rangle \end{cases}$$



Halting Problem Proof



- H accepts <M, w> exactly when M accepts <M>
- D rejects <M> exactly when M accepts <M>
- D rejects <D> exactly when D accepts <D>, which is impossible
- Similarly, D accepts <D> exactly when D rejects <D>, which is also impossible
- In both the cases, H determines the wrong answer for D
- Because this is a contradiction, D and H cannot exist
- Because H cannot exist, A_{TM} must not be decidable

Diagonalization Proof of Halting Problem

- We list all possible Turing Machines M_1, M_2, M_3, … down the rows and their descriptions <M_1>, <M_2>, <M_3>, … across the column in the paring matrix as

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	
M_4	*accept*	*accept*	*reject*	*reject*	
$\vdots$		$\vdots$			

- <i, j>th entry of the matrix says whether TM M_i in the ith row accepts or rejects the input <M_j> in the jth column

Diagonalization Proof of Halting Problem

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*		*reject*	
M_4	*accept*	*accept*	*reject*	*reject*		*accept*	
$\vdots$		$\vdots$			$\ddots$		
D	*reject*	*reject*	*accept*	*accept*		?	
$\vdots$		$\vdots$					$\ddots$

- Then we essentially construct the TM D to do the opposite of diagonal entries
- D cannot be listed on the matrix because on column (D, <D>), D cannot contain a ‘accept’ or a ‘reject’
  - If (D, <D>) is ‘reject’, then D rejects input <D>
  - Therefore by definition of H, H has to reject (D, <D>)
  - But by definition of D, D has to accept <D>, contradiction
- Similarly, if (D, <D>) is ‘accept’, then we reach a contradiction
- Therefore, D cannot exist. But, D would be easy to construct if we had H. Therefore, H cannot exist. Therefore A_{TM} is not decidable (RE, but not recursive).

Reducibility and Undecidability

- Language A is reducible to language B (represented as $A \le B$) if there exists a function *f* which will convert strings in A to strings in B as:

  $w \in A \Leftrightarrow f(w) \in B$

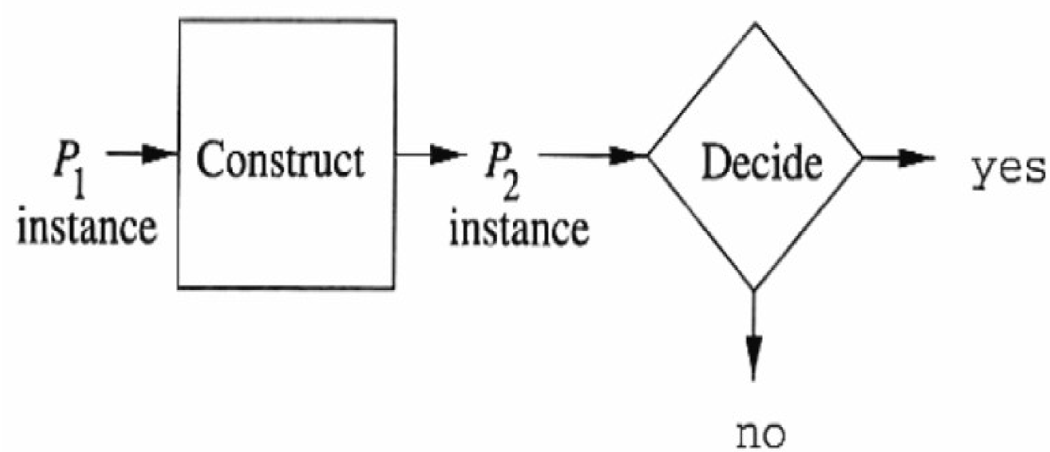
- Theorem 3: If $A \le B$ and B is recursive then A is also recursive
- Theorem 4: If $A \le B$ and A is RE, but not recursive (undecidable) then B is also RE, but not recursive
- Theorem 5: If $A \le B$ and A is Non-RE then B is also Non-RE

Usage of Reducibility

- Use a known undecidable problem as the "seed" to prove other languages are undecidable
- In each case, take a **known** undecidable language and reduce it to the **unknown** one, thereby proving that the unknown one is also undecidable
- "problem P_1 reduces to problem P_2" means that, in some sense, P_2 is equally as general as or more general than P_1 because P_2 can decide for P_1

Problem P_1 is reduced to P_2

- Take a **general** instance of problem P_1
- Find a way to transform this general instance into a **specific** instance of problem P_2 so that "solving P_2 will solve P_1"



Languages used for Reduction

1. Language of a Universal TM ("UTM")

   L_u= {<M, w> | M accepts w}

   Result: L_u is in RE, but not recursive

2. Diagonalization language

   L_d= {w_i | M_i does not accept w_i}

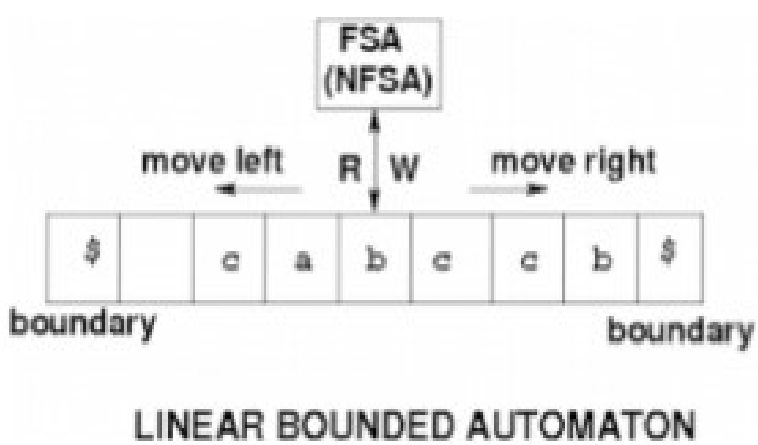
   Result: L_d is non-RE

- Example of usage of reduction in TM:

1. L_{ne}, TM that accepts the nonempty Languages (consists of all strings except empty string). To prove that is RE, but not recursive, we reduce L_u to L_{ne}
2. L_e, TM that accepts the Empty Language (consists of only empty string (TM does not accept any input)). To prove that is non-RE, we reduce L_{ne} to L_e.

Linear Bounded Automata (LBA)

- LBA is a nondeterministic TM with a bounded finite input tape
- Input is placed between two special end marker tape symbols # and $
- All actions of a standard TM are allowed except that # and $ cannot be altered and the read/write tape head cannot go on left of # and right of $



LBA (Contd)

- LBA M = (Q, ∑, Γ, δ, q_0, #, $, F) where Q, ∑, Γ, δ, q_0, F are same as that of a std TM
- Language accepted by LBA M is L(M) = {w | w ϵ ∑* and q0#w$ |-* $^{\beta}p^{\gamma}$, for some p ϵ F}
- Computations on LAB are restricted to the portion of the tape containing the input plus the two tape cells holding the end markers
- This limitation makes an LBA a somewhat more accurate model of a real-world computer than a TM, whose definition assumes unlimited tape

Context Sensitive Languages

- Language accepted by LBA is called Context Sensitive Languages (CSL)
- The only restriction placed on grammars for such language is the productions are of the form $A \rightarrow B$ where $A, B \in (V \cup T)^*$ and $|A| \leq |B|$
- Thus no derivation of a string in a context-sensitive language can contain a sentential form longer than the string itself
- Since there is a one-to-one correspondence between LBA and such grammars, no more tape than that occupied by the original string is necessary for the string to be recognized by the automaton

The Chomsky Hierarchy

- Is a containment hierarchy of classes of languages

Grammars	Languages	Accepting Machines
Type 0 grammars, phrase-structure grammars, unrestricted grammars	Recursively Enumerable languages	Turing Machine, Nondeterministic Turing Machine
Type 1 grammars, Context Sensitive Grammars,	Contexts Sensitive languages	Linear Bounded Automata
Type 2 grammars, context-free grammars	Context Free languages	Pushdown Automata
Type 3 grammars, regular grammars, left-linear grammars, right-linear grammars	Regular languages	Deterministic Finite Automata, Nondeterministic Finite Automata

Thanks