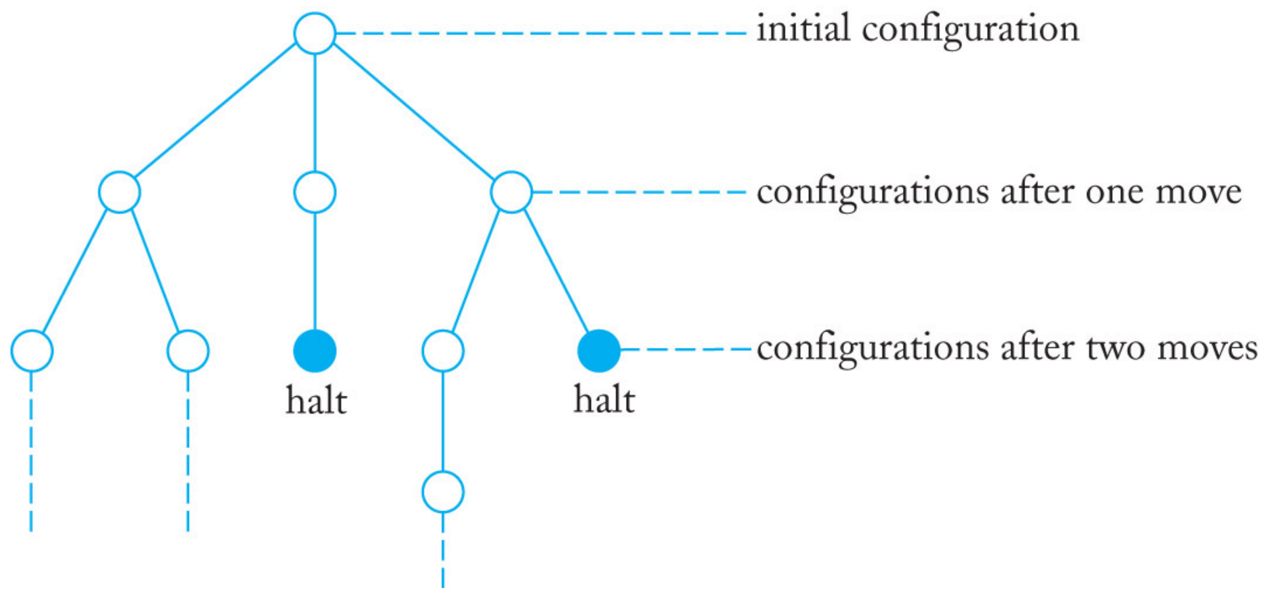
# CS 3313
# Foundations of Computing:

## 1. Review: Example of Non-deterministic algorithm
## 2. Review: Diagonalization proof technique

http://gw-cs3313.github.io

# Nondeterministic TM's - Review

- Allow the TM to have a choice of move at each step.
  - Each choice is a state-symbol-direction triple, as for the deterministic TM.
- The TM accepts its input if a sequence of choices leads to an accepting state.
- Transition function: $\delta: Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L,R\}}$
- From a configuration, i.e., ID, $\alpha p a \beta$ after one move the TM goes to another ID $\alpha c q_i \beta$ if $\delta(p, a) = (q_i, c, R)$ is contained in $\delta(p, a)$
  - $(q_i, c, R)$ is one of the choices of moves the machine can make from $(p,a)$
- Machine starts in ID $q_0 w$



# Simulation of a NDTM on a TM

- Theorem: For any Non-deterministic Turing Machine, there is an equivalent deterministic Turing Machine.
  - If a language is accepted (or a function is computed) by a non-deterministic TM then there is a deterministic turing machine that accepts that language (or computes the function).
  - *NDTM and TM are equally powerful and solve the same class of problems !*
- Multi-tape Deterministic TM can simulate a non-deterministic TM
- Simulation time: If NDTM accepts in $n$ moves, and it has at most $k$ choices from each configuration, then Deterministic TM accepts in $O(k^n)$ moves
  - *Polynomial time on NDTM but exponential time on DTM*

# The classes P and NP

- A problem is in class NP if there is a polynomial time non-deterministic algorithm to solve the problem
- A problem is in class P if there is a polynomial time deterministic algorithm to solve the problem
- P = NP ?  Open problem….(our simulation suggests not equal)
- NP-Complete: A problem is NPC if it can be reduced to the SAT problem….
  - If this problem can be solved then all problems in NPC can be solved in polynomial deterministic time

# Solvable/Decidable vs Unsolvable/Undecidable problems

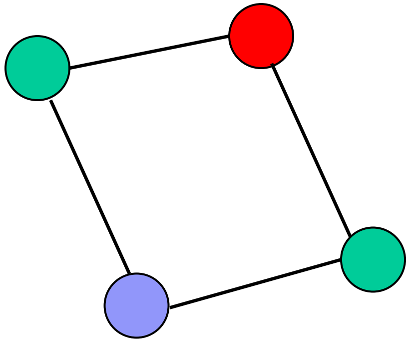
- Algorithm = Turing machine that halts on all inputs (always halts)
  - Decision problem: the answer is "Yes" or "No"
- A problem is undecidable if there is no algorithm (Turing machine that always halts) that solves the problem
  - Problem = language
  - How do we show a problem is undecidable/unsolvable – need to prove the problem is undecidable

- A problem is decidable if there is an algorithm (Turing machine that always halt) to solve the problem
  - How do we show a problem is solvable – provide an algorithm that solves the problem
  - *Key observation: the algorithm can be deterministic or non-deterministic when we are trying to prove it is solvable/decidable*

# Graph Coloring is a solvable problem

- Graph Coloring: Given a graph $G=(V,E)$, find the minimum number of colors $k$ such that no two adjacent (connected) vertices have the same color
  - Recall: definition and properties from Discrete 2 !!
  - Applications: include allocating registers to variables in the program (by the compiler)…the register allocation problem

Valid coloring with 3 colors



# A Non-deterministic Algorithm for Graph Coloring

- Non-determinism is a powerful expressive tool to design solutions
  - Implementation on a computer requires deterministic algorithms
- Example: Graph Coloring Problem
  - Known to be NP-complete.
  - How can we solve it in polynomial time using a non-deterministic algorithm
- We outline a simple (there are better solutions) polynomial time non-deterministic algorithm

# Non-deterministic Algorithm for Graph Coloring (1)

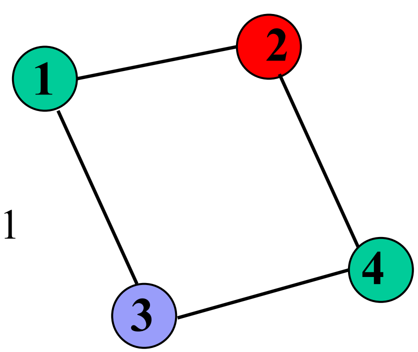
- First Step is design `Function Check_Coloring(G,C):` given a mapping of colors to vertices, check if it is a valid coloring of the graph.
  - Let $V = \{v_1, v_2 \ldots, v_n\}$
- A coloring C assigns one of k colors to each vertex,

$$C: \{\, 1,\ldots,k\} \rightarrow \{v_1, v_2 \ldots, v_n\}$$

- Is $C_1$ a valid coloring of $G_1$ ? Is $C_2$ a valid coloring of $G_2$

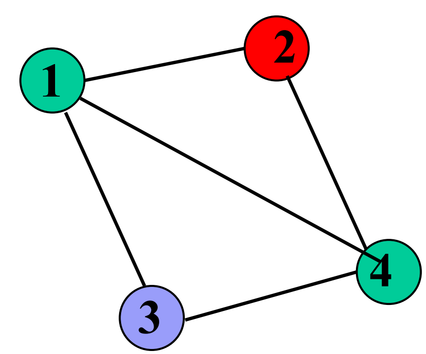
Coloring $C_1$:
C(1)= green, C(2)- red,
C(3)= blue C(4)=green

Graph $G_1$




Coloring $C_2$:
C(1)= green, C(2)- red,
C(3)= blue C(4)=green

Graph $G_1$



# Non-deterministic Algorithm for Graph Coloring (1)

- A coloring C assigns one of k colors to each vertex,

$$C: \{ 1,..,k\} \rightarrow \{v_1, v_2 ..., v_n\}$$

- `Function Check-Coloring(G,C)`
  - `Input is G=(V,E) and a coloring C`
  - `For each vertex` $v_i$ `check if the coloring is valid`
    - `Check if` $C(v_i)$ `<>` $C(v_j)$ `for all` $v_j$ `where` $(v_i, v_j)$ *in E* `(edge between` $v_i$ `and` $v_j$`)`
    - `If valid then return “Yes”`
- This is a deterministic algo…takes $O(n^2)$ (examine all edges)
  - For each of the $n$ vertices $v_i$ , check all edges (max no. edges is $O(n^2)$)

# Non-deterministic Algorithm for Graph Coloring (1)

- A coloring C assigns one of k colors to each vertex,

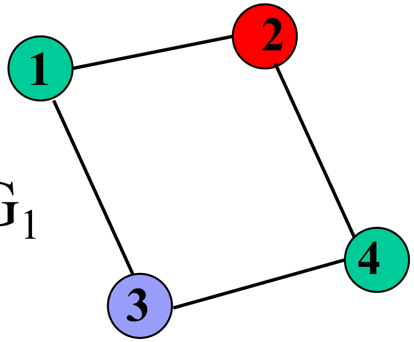
$$C\colon \{ 1,..,k\} \rightarrow \{v_1,v_2...,v_n\}$$

- Next: generate a coloring C using *k* colors, starting with *k=1*
- How to represent a coloring: a string of length *n* where each symbol is from the set *{1,2,…k}* ! *(yep – enumeration once again !)*
  - Ex: For *n=7* and *k=4* the string **2134142** is a coloring (may not be valid)
  - $C(v_1)= 2$, $C(v_2)= 1$, $C(v_3)= 3$, $C(v_4)= 4$, $C(v_5)= 1$, $C(v_6)= 4$, $C(v_7)= 2$

- Example: In $G_1$ the coloring is the sequence 1231

where 1=Green, 2=Red, 3=Blue

Graph $G_1$



# Non-deterministic Algo for Coloring – (2)

- Next: generate a coloring C using $k$ colors, starting with $k=1$
- Given $n$ (number of vertices in graph) and $k$ (number of colors), we can generate all strings of length $n$ – i.e., all $n$ ‘digit’ base k numbers
- How ?....discussion of algorithm from lecture
  - Generate in lexicographic order (or smallest numeric value first)
- For each such coloring (base $k$ number of length $n$) *branch non-deterministically* and call function Check_Coloring

# Non-deterministic Algo for Coloring – (3)

1. done= False   k=1
2. While (not done) {
   1. Generate all colorings $C_j$ of length n using k colors(/symbols)
   2. For each coloring $C_j$ simultaneously (non-deterministically) start function Check_Coloring($C_j$,G)
   - If any of these return "Yes" then done=True else k= k+1
3. Return k (minimum number of colors needed)

- Note that the algorithm will eventually halt – since we know that $k=n$ is a valid coloring (i.e., each vertex colored with its own color)
- Note: for $n$ and $k$, there are $k^n$ different colorings that we have to check
  - One concurrent branch in a NDTM…but no concurrent branches in DTM!!

# Questions ?

# More Set theory review…..Today

1. Quick recall from last week lab: Concept of Countable and Uncountable sets
2. Today: use diagonalization method to prove set of Real numbers is uncountable

# Countable and Uncountable Sets

- Set cardinality: number of elements in the set (size of the set)
- Intuition: if we can arrange the elements of set in a manner where we can speak of "first element", "second element",etc.

- An infinite set A is countably infinite *if and only if* it has the same cardinality as the set of Natural numbers (positive integers)
  - There is a one to one correspondence (one to one and onto) from A to $N$.
- A set is countable *iff* it is finite or is countably infinite
- A set that is not countable is said to be uncountable
- Useful Theorems:
  - 1. If $A \subseteq B$ and B is countable then A is countable
  - 2. If $A \subseteq B$ and A is uncountable then B is uncountable

# Countably Infinite Sets…Example

- Two sets have the same cardinality if there is a one to one correspondence between them
- An *enumeration* of a set is a 1-1 correspondence between the set and the positive integers
- Set of all integers $Z$ is a countably infinite set: $f$: $Z$ -> $N$
  - $f(0) = 1$ $f(-i)= 2i$ $f(+i)= 2i+1$
  - *0 -> 1, -1 -> 2, 1 -> 3, -2 -> 4,…*
  - *"ordering" 0, -1, 1, -2, 2, -3, 3, …..*
- Set of even integers $Z_2$ is countably infinite $f_2$: $N$ -> $Z_2$
  - $f_2 (n) = 2n$
- Set of primes $P$ is countably infinite – $f_3$
  - $f_3$: *(p) : p is the i-th prime.*
    - Recall: we proved earlier that set of primes is infinite

# Rational Numbers $Q$

- Rational number $p/q$          $p,q$ are integers

Last week lab we proved:

- **Theorem:** Set of positive rational numbers Q is countable
- Proof:
- Intuition: list the rational numbers "in order"
  - Find a way to "label" the rational numbers to get the first rational number, the second rational number, etc.
    - For simplicity, let's work with $p,q$ positive
  - Observe: We can view the number $p/q$ as a pair of integers *[p,q]* and then order them first by sum and then by first component
    - [1,1], [2,1], [1,2], [3,1], [2,2], [1,3], [4,1], [3,2],..[1,4],[5,1]…

# Ordering of (positive) Rational Numbers $Q$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1/1 | 1/2 | 1/3 | 1/4 | 1/5 | … | … | .. |
| 2/1 | 2/2 | 2/3 | 2/4 | 2/5 | … | … | … |
| 3/1 | 3/2 | 3/3 | 3/4 | 3/5 | … | … | … |
| 4/1 | 4/2 | 4/3 | 4/4 | 4/5 | … | … | … |
| 5/1 | 5/2 | 5/3 | 5/4 | 5/5 | … | … | … |
| …. | … | … | | | | … | … |
| … | … | … | | | | … | … |
| … | … | | | | | | |

# Key Idea today – Diagonalization

- Generate a table to describe property of collection (set) of objects
- Then manipulate the ”diagonal” in the table to get a new object that you can prove is ***not*** in the table

# Set of Real Numbers *R*: An uncountable set

- A real number has a decimal representation
  - $\pi = 3.1415926.....$
  - $\sqrt{2} = 1.4142135...$
- A trick similar to rationals does not work here….but will use it to develop a proof by contradiction
- Theorem: Set of real number *R* is uncountable
- Proof by contradiction….
  - The technique, developed by Cantor, is known as *diagonalization*
- *We will prove that the set of reals between 0 and 1 (0,1) is uncountable*
  - This is a subset of R, therefore R is uncountable

# Uncountable Set.....concept of Diagonalization

- Theorem: Set of real numbers is uncountable.
- Proof by contradiction: Assume the set (0,1) is countable.
- Any number in this set can be represented as $0.d_1d_2d_3....$
- Suppose the set is countable, then we can write a list of real numbers and count them from *1 to n*: $x_1, x_2, ...x_n, ..$
  - *Each* $x_i = 0.d_1d_2.....$ *Where* $d_i$ *is a digit*

# Uncountable Set…..concept of Diagonalization

- Any number in this set (0,1) can be represented as $0.d_1d_2d_3$….
- Suppose the set is countable, then we can write a list of real numbers and count them from *1 to n*: $x_1, x_2, \ldots x_n, \ldots$
  - *Each* $x_i = 0.d_1d_2\ldots\ldots$ *Where* $d_i$ *is a digit*
- Notation: for each number $x_i$ in the list (this appears in the *i-th* position) we can list the values of the digits $a_{ij}$ in each position $j$ after the decimal

$x_1 = 0.\ a_{11}\, a_{12}\, a_{13}\, a_{14}\ldots$ Ex: $x_1 = 0.1342..$ then $a_{11}=1,\ a_{12}=3,\ a_{13}=4..$

$x_2 = 0.\ a_{21}\, a_{22}\, a_{23}\, a_{24}\ldots$

..

$x_i = 0.a_{i1}\, a_{i2}\, a_{i3}\ldots.a_{ij}$

# Uncountable set: (0,1) of reals

- Now view this concept as a matrix
  - Infinite number of columns and rows
  - *i-th* row is real number $x_i$
  - *j-th* column is value of $a_{ij}$ – the value in *j-th* decimal position of $x_i$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $a_{11}$ | $a_{12}$ | $a_{13}$ | $a_{14}$ | $a_{14}$ | … | … | .. |
| $a_{21}$ | $a_{22}$ | $a_{23}$ | $a_{24}$ | $a_{25}$ | … | … | … |
| $a_{31}$ | $a_{32}$ | $a_{33}$ | $a_{34}$ | $a_{35}$ | … | … | … |
| $a_{41}$ | $a_{42}$ | $a_{43}$ | $a_{44}$ | $a_{45}$ | … | … | … |
| … | … | … | … | … | … | … | … |
| $a_{i1}$ | $a_{i2}$ | $a_{i3}$ | | | | $a_{ij}$ | … |
| … | … | … | | | | … | … |
| … | … | | | | | | |

# Example

- Illustrate construction (of matrix) with an example..
- $x_1 = 0.23117\ldots.$ $x_2=0.11654\ldots$ $x_3=0.14142..$ $x_4=0.40375\ldots$

$$\xrightarrow{\quad j \quad}$$

| $i \downarrow$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $x_1$ | 2 | 3 | 1 | 1 | 7 | … | … | .. |
| $x_2$ | 1 | 1 | 6 | 5 | 4 | … | … | … |
| | 1 | 4 | 1 | 4 | 2 | … | … | … |
| | 4 | 0 | 3 | 7 | 5 | … | … | … |
| | … | … | … | … | … | … | … | … |
| | $a_{i1}$ | $a_{i2}$ | $a_{i3}$ | | | | $a_{ij}$ | … |
| | … | … | … | | | | … | … |
| | … | … | | | | | | |

# Example

Consider the entries on the diagonal $a_{ii}$ and construct $y$ such that $y$ is not in this (infinite) matrix – **contradiction** since we said every real number in (0,1) can be listed in this manner.

$j \longrightarrow$

$i \downarrow$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *2* | *3* | *1* | *1* | *7* | … | … | .. |
| *1* | *1* | *6* | *5* | *4* | … | … | … |
| *1* | *4* | *1* | *4* | *2* | … | … | … |
| *4* | *0* | *3* | *7* | *5* | … | … | … |
| … | … | … | … | … | … | … | … |
| $a_{i1}$ | $a_{i2}$ | $a_{i3}$ | | | | $a_{ij}$ | … |
| … | … | … | | | | … | … |
| … | … | | | | | | |

- $x_1 = 0.23117....$ $x_2 = 0.11654...$ $x_3 = 0.14142..$ $x_4 = 0.40375...$

# Example- Contradiction

If $y$ is listed in this matrix then $y= x_k$, for some $k$, and $x_k= 0.a_{k1}\, a_{k2}\, a_{k3}..\, a_{kj}\ldots$

***Pick y such that $a_{kj} \neq a_{jj}$ for all $j$***

How? Ex: define $a_{kj} = 2$ if $a_{jj} = 1$ else $a_{kj} = 1$

***ex: $y= 0.1221\ldots$***

$j \longrightarrow$

$i \downarrow$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 **1** | 3 | 1 | 1 | 7 | … | … | .. |
| 1 | 1 **2** | 6 | 5 | 4 | … | … | … |
| 1 | 4 | 1 **2** | 4 | 2 | … | … | … |
| 4 | 0 | 3 | 7 **1** | 5 | … | … | … |
| … | … | … | … | … | … | … | … |
| $a_{i1}$ | $a_{i2}$ | $a_{i3}$ | | | | $a_{ij}$ | … |
| … | … | … | | | | … | … |
| … | … | | | | | | |

- $x_1= 0.23117\ldots.\ x_2=0.11654\ldots\ \ x_3=0.14142..\ \ x_4=0.40375\ldots$

# Example- Contradiction

If $y$ is listed in this matrix then $y = x_k$, for some $k$, and $x_k = 0.a_{k1}\, a_{k2}\, a_{k3} .. \, a_{kj} \ldots$

***Pick y such that $a_{kj} \neq a_{jj}$ for all $j$***

How? Ex: define $a_{kj} = 2$ if $a_{jj} = 1$ else $a_{kj} = 1$

***ex: $y = 0.1221\ldots$***

$j$ →

$i$ ↓

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *2* **1** | *3* | *1* | *1* | *7* | … | … | .. |
| *1* | *1* **2** | *6* | *5* | *4* | … | … | … |
| *1* | *4* | *1* **2** | *4* | *2* | … | … | … |
| *4* | *0* | *3* | *7* **1** | *5* | … | … | … |
| … | … | … | … | | … | … | … |
| $a_{i1}$ | $a_{i2}$ | $a_{i3}$ | | | | $a_{ij}$ | … |
| … | … | … | | | | … | … |
| … | … | | | | | | |

0.1221… cannot be in the matrix:
cannot be in row 1
cannot be in row 2
cannot be in row 3
etc….

- $x_1 = 0.23117 \ldots .\ x_2 = 0.11654 \ldots\ x_3 = 0.14142 ..\ x_4 = 0.40375 \ldots$

# Proof - contradiction

- Consider the number y, where y= $0.a_{k1}\, a_{k2}\, a_{k3}..\, a_{kj}...$ *where* $\boldsymbol{a_{kj} \neq a_{jj}}$ ***for all j***
- As one instance: pick $a_{kj} = 2$ *if* $a_{jj} = 1$ else $a_{kj} = 1$
- **Claim**: this number $y$ cannot appear in the matrix as any $x_k$
- Proof: if $y = x_k$ for some $k$, then we have at least one decimal position $j$ where $a_{kj}$ is not equal to value in row $k$, column $j$ in the matrix. Contradiction
  - Example: ***y= 0.1221…***
    - *Cannot be in the matrix…cannot be in any row because at least one column j (the diagonal entry) the value is not the same as value in the complete matrix.*

This type of proof construction – diagonalization is due to Cantor

# Another example of an uncountable set: How Many Languages?

- Are the languages over $\{0,1\}$ countable?
  - Recall: A language over $\{0,1\}$ is a subset of $\{0,1\}^*$
    - Set of strings over $\{0,1\}$
- No – here's a proof.
- Suppose we could enumerate all languages over $\{0,1\}$ and talk about "the *i-th* language."
- Consider the language $L = \{ w \mid w$ is the *i-th* binary string and $w$ is not in the *i-th* language$\}$.
  - We discussed (last week's lab) one way to enumerate binary strings
    - So we can talk about the *i-th* binary string

# Proof – Continued

- Clearly, $L$ is a language over $\{0,1\}$.
- Thus, it is the *j-th* language for some particular $j$.
- Let $x$ be the *j-th* string.
- Is $x$ *in* $L$?
  - If so, $x$ is not in $L$ by definition of $L$.
  - If not, then $x$ is in $L$ by definition of $L$.

$L_j$

$x$

Recall: $L = \{\, w \mid w$ is the *i-th* binary string and $w$ is not in the *i-th* language$\}$.

*j-th*

# Diagonalization Picture

Imagine an in finite matrix:

rows correspond to languages and columns to strings

a *1* in the entry for row *i*, column *j* means *j-th* string is in the *i-th* language

If we could enumerate languages, we could create such a table

Strings

| Languages | 1 | 2 | 3 | 4 | 5 | … |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 0 | … |
| 2 | | 1 | | | | |
| 3 | | | 0 | | | |
| 4 | | | | 0 | | |
| 5 | | | | | 1 | |
| … | | | | | | … |

# Diagonalization Picture

$L = \{ w \mid w$ is the *i-th* binary string and $w$ is not in the *i-th* language$\}$.

To construct L:
Flip each
diagonal
entry

Strings

| Languages | 1 | 2 | 3 | 4 | 5 | … |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 0 | … |
| 2 | | 0 | | | | |
| 3 | | | 1 | | | |
| 4 | | | | 1 | | |
| 5 | | | | | 0 | |
| … | | | | | | … |

Can’t be
a row –
it disagrees
in an entry
of each row.
Disagrees with
*i-th* row in
*i-th* position

# Proof – Concluded

- We have a contradiction: $x$ is neither in $L$ nor not in $L$, so our sole assumption (that there was an enumeration of the languages) is wrong.
- Comment: *This is really bad; there are more languages than programs.*
- E.g., there are languages with no membership algorithm.

*Why is this proof important:*
*this proof is used to provide a proof of a problem that cannot be solved by a turing machine !!*