



# UNIT - V

## Types of Turing machine

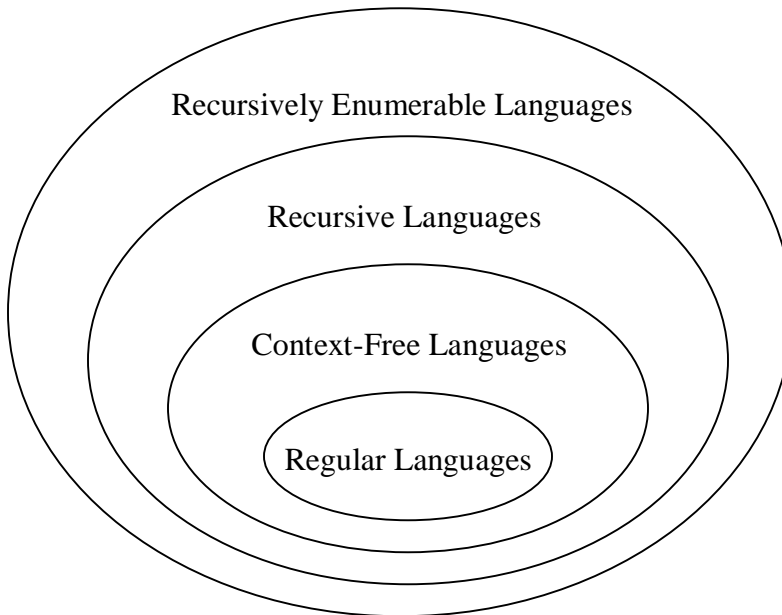
Turing machines and halting

## Undecidability

Undecidability, A Language that is Not Recursively Enumerable, An Undecidable Problem That is RE, Undecidable Problems about Turing Machines, Recursive languages, Properties of recursive languages, Post's Correspondence Problem, Modified Post Correspondence problem, Other Undecidable Problems, Counter machines.

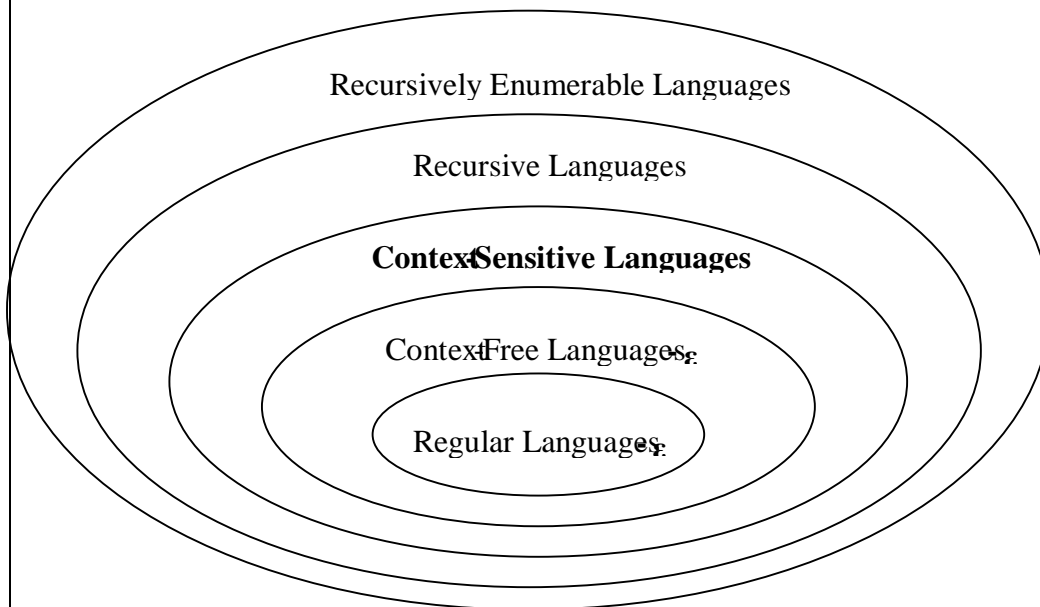
## Turing Machines (TM)

- Generalize the class of CFLs:  
Non-Recursively Enumerable Languages



- Another Part of the Hierarchy:**

Non-Recursively Enumerable Languages



- Recursively enumerable languages are also known as *type 0* languages.
- Context-sensitive languages are also known as *type 1* languages.
- Context-free languages are also known as *type 2* languages.
- Regular languages are also known as *type 3* languages.
- TMs model the computing capability of a general purpose computer, which informally can be described as:
  - Effective procedure
- Finitely describable • Well defined, discrete, “mechanical” steps
- Always terminates
  - Computable function
- A function computable by an effective procedure
- TMs formalize the above notion.
- **Church-Turing Thesis:** There is an effective procedure for solving a problem if and only if there is a TM that halts for all inputs and solves the problem.
  - There are many other computing models, but all are equivalent to or subsumed by TMs. *There is no more powerful machine* (Technically cannot be proved).
- DFAs and PDAs do not model all effective procedures or computable functions, but only a subset.

*Deterministic Turing Machine (DTM)*



Finite  
Control

- Two-way, infinite tape, broken into cells, each containing one symbol.
- Two-way, read/write tape head.
- An input string is placed on the tape, padded to the left and right infinitely with blanks, read/write head is positioned at the left end of input string.
- Finite control, i.e., a program, containing the position of the read head, current symbol being scanned, and the current state.
- In one move, depending on the current state and the current symbol being scanned, the TM 1) changes state, 2) prints a symbol over the cell being scanned, and 3) moves its' tape head one cell left or right.
- Many modifications possible, but Church-Turing declares equivalence of all.

#### *Formal Definition of a DTM*

- A DTM is a seven-tuple:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

$Q$       A finite set of states

$\Sigma$       A finite input alphabet, which is a subset of  $\Gamma - \{B\}$

$\Gamma$       A finite tape alphabet, which is a strict superset of  $\Sigma$

B A distinguished blank symbol, which is in  $\Gamma$   $q_0$  The  
initial/starting state,  $q_0$  is in  $Q$

F A set of final/accepting states, which is a subset of  $Q$

$\delta$  A next-move function, which is a *mapping* (i.e., may be undefined)

from

$$Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

Intuitively,  $\delta(q, s)$  specifies the next state, symbol to be written, and the direction of tape head movement by  $M$  after reading symbol  $s$  while in state  $q$ .

- **Example #1:**  $\{w \mid w \text{ is in } \{0,1\}^* \text{ and } w \text{ ends with a } 0\}$

0

00

10

10110

Not  $\varepsilon$

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

$$\Gamma = \{0, 1, B\}$$

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

$$F = \{q_2\}$$

$\delta$ :

	0	1	B
$\rightarrow q_0$	$(q_0, 0, R)$	$(q_0, 1, R)$	$(q_1, B, L)$
$q_1$	$(q_2, 0, R)$	-	-
$q_2^*$	-	-	-

- $q_0$  is the start state and the “scan right” state, until hits B

- $q_1$  is the verify 0 state
- $q_2$  is the final state
- **Example #2:**  $\{0^n 1^n \mid n \geq 1\}$

	0	1	X	Y
-> $q_0$	( $q_1$ , X, R)	-	-	( $q_3$ , Y, R) <i>0's</i> - <b>finished</b>
$q_1$	( $q_1$ , 0, R) <b>ignore1</b>	( $q_2$ , Y, L)	-	( $q_1$ , Y, R) - (more 0's)
$q_2$	( $q_2$ , 0, L) <b>ignore2</b>	-	( $q_0$ , X, R)	( $q_2$ , Y, L) -
$q_3$	-	- (more 1's)	-	( $q_3$ , Y, R) (q4, B, R)
$q_4$	-	-	-	-
*	-	-	-	-

- **Sample Computation:** (on 0011), *presume state  $q$  looks rightward*
- Q00011BB.. |—
- X
- $q_1$ 011|—
- X0 $q_1$ 11|—
- X0 $q_1$ X<sub>2</sub>0Y1|—
- $q_2$  X0 Y1

|— X $q_0$ 0Y1

|— XX $q_1$ Y1

|— XXYq<sub>1</sub>1  
 |— XXq<sub>2</sub>YY  
 |— Xq<sub>2</sub>XYX  
 |— XXq<sub>0</sub>YY  
 |— XXY q<sub>3</sub> B...  
 |— XXYY q<sub>3</sub>B B...  
 |— XXYYBq<sub>4</sub>

Making a TM for  $\{0^n 1^n \mid n \geq 1\}$

Try  $n=2$  or  $3$  first.

- q<sub>0</sub> is on 0, replaces with the character to X, changes state to q<sub>1</sub>, moves right
- q<sub>1</sub> sees next 0, ignores (both 0's and X's) and keeps moving right
- q<sub>1</sub> hits a 1, replaces it with Y, state to q<sub>2</sub>, moves left
- q<sub>2</sub> sees a Y or 0, ignores, continues left
- when q<sub>2</sub> sees X, moves right, returns to q<sub>0</sub> for looping step 1 through 5
- when finished, q<sub>0</sub> sees Y (no more 0's), changes to pre-final state q<sub>3</sub>
- q<sub>3</sub> scans over all Y's to ensure there is no extra 1 at the end (to crash on seeing any 0 or 1)
- when q<sub>3</sub> sees B, all 0's matched 1's, done, changes to final state q<sub>4</sub> • blank line for final state q<sub>4</sub>

Try  $n=1$  next.

Make sure unbalanced 0's and 1's, or mixture of 0-1's,

“crashes” in a state not  $q_4$ , as it should be

$q_0 0011BB..$  |—

$Xq_1 011$

|—

$X0q_1 11$

|—

$Xq_2 0Y1$

|—

$q_2 X0Y1$

|—

$Xq_0 0Y1$

|—  $XXq_1 Y1$

|—  $XXYq_1 1$

|—  $XXq_2 YY$

|—  $Xq_2 XYY$

|—  $XXq_0 YY$

|—  $XXYq_3 Y$

|—  $Bq_4 B$

$XXYYq_3$

$BB...$

|—

$XXYYBq_4$

• S  
ame  
Exam  
ple  
#2:  
 $\{0^n 1^n \mid n \geq 1\}$

$(q_1, X, R)$

-

-

$(q_3, Y, R)$

-

$(q_1, 0, R)$

$(q_2, Y, L)$

-

$(q_1, Y, R)$

-

$(q_2, 0, L)$

-

$(q_0, X, R)$

$(q_2, Y, L)$

-

-

-

-

$(q_3, Y, R)$

-



q <sub>0</sub>	-	-	-	-	-
q <sub>1</sub> q <sub>2</sub> q <sub>3</sub> q <sub>4</sub>					

*Logic:* cross 0's with X's, scan right to look for corresponding 1, on finding it cross it with Y, and scan left to find next leftmost 0, keep iterating until no more 0's, then scan right looking for B.

- The TM matches up 0's and 1's
- q<sub>1</sub> is the "scan right" state, looking for 1
- q<sub>2</sub> is the "scan left" state, looking for X
- q<sub>3</sub> is "scan right", looking for B
- q<sub>4</sub> is the final state

*Can you extend the machine to include  $n=0$ ?*

*How does the input-tape look like for string epsilon?*

- **Other Examples:**

000111	00
11	001
011	

- **Roger Ballard's TM for Example #2, without any extra Tape Symbol:  $\{0^n 1^n \mid n \geq 0\}$**

	0	1	B
q <sub>0</sub>	(q <sub>1</sub> , B, R)		(q <sub>4</sub> , B, R)
q <sub>1</sub>	(q <sub>1</sub> , 0, R)	(q <sub>1</sub> , 1, R)	(q <sub>2</sub> , B, L)
q <sub>2</sub>	-	(q <sub>3</sub> , B, L)	-
q <sub>3</sub>	(q <sub>3</sub> , 0, L)	(q <sub>3</sub> , 1, L)	(q <sub>0</sub> , B, R)
q <sub>4</sub> *	-	-	-

*Logic:* Keep deleting 0 and corresponding 1 from extreme ends, until none left.

- $q_0$  deletes a leftmost 0 and let  $q_1$  scan through end of string,  $q_0$  accepts on epsilon
- $q_1$  scans over the string and makes  $q_2$  expecting 1 on the left
- $q_2$  deletes 1 and let  $q_3$  “scan left” looking for the start of current string
- $q_3$  lets  $q_0$  start the next iteration
- $q_4$  is the final state

*Any bug?*

Try on:

000111	00
11	001
011	

- **And his example of a correct TM for the language that goes on infinite loop outside language:**  $\{0^n 1^n \mid n \geq 0\}$

	0	1	B
$q_0$	$(q_1, B, R)$	$(q_3, 1, L)$	$(q_4, B, R)$
$q_1$	$(q_1, 0, R)$	$(q_1, 1, R)$	$(q_2, B, L)$
$q_2$	-	$(q_3, B, L)$	-
$q_3$	$(q_3, 0, L)$	$(q_3, 1, L)$	$(q_0, B, R)$
$q_4^*$	-	-	-

*Logic:* This machine still works correctly for all strings in the language, but start a string with 1 (not in the language), and it loops on B1 forever.

- **Exercises:** Construct a DTM for each of the following.
  - $\{w \mid w \text{ is in } \{0,1\}^* \text{ and } w \text{ ends in } 00\}$
  - $\{w \mid w \text{ is in } \{0,1\}^* \text{ and } w \text{ contains at least two } 0\text{'s}\}$
  - $\{w \mid w \text{ is in } \{0,1\}^* \text{ and } w \text{ contains at least one } 0 \text{ and one } 1\}$
  - Just about anything else (simple) you can think of

*Formal Definitions for DTMs*

- Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$  be a TM.
- **Definition:** An *instantaneous description* (ID) is a triple  $\alpha_1 q \alpha_2$ , where:
  - $q$ , the current state, is in  $Q$
  - $\alpha_1 \alpha_2$ , is in  $\Gamma^*$ , and is the current tape contents up to the rightmost non-blank symbol, or the symbol to the left of the tape head, whichever is rightmost
  - The tape head is currently scanning the first symbol of  $\alpha_2$
  - At the start of a computation  $\alpha_1 = \epsilon$
  - If  $\alpha_2 = \epsilon$  then a blank is being scanned

- **Example:** (for TM #1)

$q_0 0 0 1 1$      $X q_1 0 1 1$      $X 0 q_1 1 1$      $X q_2 0 Y 1$      $q_2 X 0 Y 1$   
 $X q_0 0 Y 1$      $XX q_1 Y 1$      $XX Y q_1 1$      $XX q_2 Y Y$      $X q_2 X Y Y$   
 $XX q_0 Y Y$      $XX Y q_3 Y$      $XX Y Y q_3$      $XX Y Y B q_4$

- Suppose the following is the current ID of a DTM

$x_1 x_2 \dots x_{i-1} q x_{i+1} \dots x_n$

Case 1)  $\delta(q, x_i) = (p, y, L)$

- (a) if  $i = 1$  then  $q x_1 x_2 \dots x_{i-1} x_{i+1} \dots x_n \vdash p B y x_2 \dots x_{i-1} x_{i+1} \dots x_n$

(b) else  $x_1x_2\dots x_{i-1}qx_{i+1}\dots x_n \mid\!\!\!-\ x_1x_2\dots x_{i-2}p x_{i-1}y_{x_{i+1}}\dots x_n$

- If any suffix of  $x_{i-1}y_{x_{i+1}}\dots x_n$  is blank then it is deleted.

Case 2)  $\delta(q, x_i) = (p, y, R)$

$x_1x_2\dots x_{i-1}qx_{i+1}\dots x_n \mid\!\!\!-\ x_1x_2\dots x_{i-1}y p x_{i+1}\dots x_n$

- If  $i > n$  then the ID increases in length by 1 symbol

$x_1x_2\dots x_nq \mid\!\!\!-\ x_1x_2\dots x_nyp$

- **Definition:** Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$  be a TM, and let  $w$  be a string in  $\Sigma^*$ . Then  $w$  is *accepted* by  $M$  iff

$q_0w \mid\!\!\!-\alpha_1p\alpha_2$

where  $p$  is in  $F$  and  $\alpha_1$  and  $\alpha_2$  are in  $\Gamma^*$

- **Definition:** Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$  be a TM. The *language accepted by  $M$* , denoted  $L(M)$ , is the set

$\{w \mid w \text{ is in } \Sigma^* \text{ and } w \text{ is accepted by } M\}$

- **Notes:**

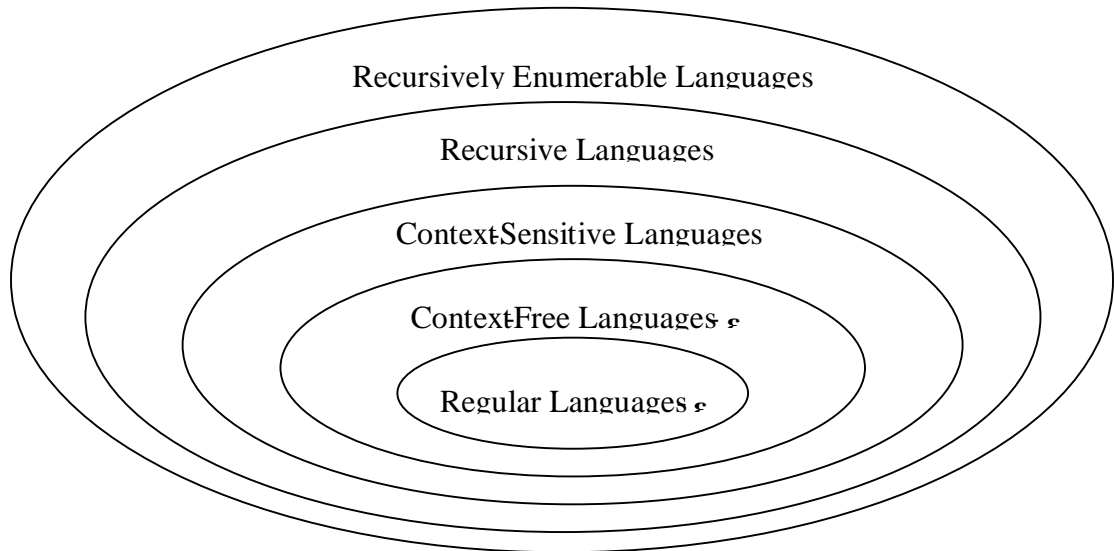
- In contrast to FA and PDAs, if a TM simply *passes through* a final state then the string is accepted.

- Given the above definition, no final state of a TM need to have any transitions. *Henceforth, this is our assumption.*
- **If  $x$  is NOT in  $L(M)$  then  $M$  may enter an infinite loop, or halt in a non-final state.**
- Some TMs halt on ALL inputs, while others may not. In either case the language defined by TM is still well defined.
- **Definition:** Let  $L$  be a language. Then  $L$  is *recursively enumerable* if there exists a TM  $M$  such that  $L = L(M)$ .
  - If  $L$  is r.e. then  $L = L(M)$  for some TM  $M$ , and
- If  $x$  is in  $L$  then  $M$  halts in a final (accepting) state.
- If  $x$  is not in  $L$  then  $M$  may halt in a non-final (non-accepting) state or no transition is available, or loop forever.
- **Definition:** Let  $L$  be a language. Then  $L$  is *recursive* if there exists a TM  $M$  such that  $L = L(M)$  and  $M$  halts on all inputs.
  - If  $L$  is recursive then  $L = L(M)$  for some TM  $M$ , and
- If  $x$  is in  $L$  then  $M$  halts in a final (accepting) state.
- If  $x$  is not in  $L$  then  $M$  halts in a non-final (non-accepting) state or no transition is available (does not go to infinite loop).

### Notes:

- The set of all recursive languages is a subset of the set of all recursively enumerable languages
- Terminology is easy to confuse: A *TM* is not recursive or recursively enumerable, rather a *language* is recursive or recursively enumerable.
- **Recall the Hierarchy:**

## Non-Recursively Enumerable Languages



- **Observation:** Let  $L$  be an r.e. language. Then there is an infinite list  $M_0, M_1, \dots$  of TMs such that  $L = L(M_i)$ .
- **Question:** Let  $L$  be a **recursive** language, and  $M_0, M_1, \dots$  a list of all TMs such that  $L = L(M_i)$ , and choose any  $i \geq 0$ . Does  $M_i$  always halt?
- **Answer:** Maybe, maybe not, but *at least one in the list does*.
- **Question:** Let  $L$  be a **recursive enumerable** language, and  $M_0, M_1, \dots$  a list of all TMs such that  $L = L(M_i)$ , and choose any  $i \geq 0$ . Does  $M_i$  always halt?
- **Answer:** Maybe, maybe not. Depending on  $L$ , none might halt or some may halt.
  - If  $L$  is also recursive then  $L$  is recursively enumerable, *recursive is subset of r.e.*

- **Question:** Let  $L$  be a r.e. language that is not recursive ( $L$  is in r.e. – r), and  $M_0, M_1, \dots$  a list of all TMs such that  $L = L(M_i)$ , and choose any  $i \geq 0$ . Does  $M_i$  always halt?

- **Answer:** No! If it did, then  $L$  would not be in r.e. – r, it would be recursive.

$L$  is Recursively enumerable:

*TM exist:  $M_0, M_1, \dots$*

*They accept string in  $L$ , and do not accept any string outside  $L$*

$L$  is Recursive: *at least one TM halts on  $L$  and on  $\Sigma^* - L$ , others may*

*or may not*  $L$  is Recursively enumerable but not Recursive:

*TM exist:  $M_0, M_1, \dots$  but none halts on all  $x$  in  $\Sigma^* - L$*

*$M_0$  goes on infinite loop on a string  $p$  in  $\Sigma^* - L$ , while  $M_1$  on  $q$  in  $\Sigma^* - L$*

*However, each correct TM accepts each string in  $L$ , and none in  $\Sigma^* - L$*

$L$  is not R.E: no TM exists

- **Let  $M$  be a TM.**

– Question: Is  $L(M)$  r.e.?

– Answer: Yes! By definition it is!

– Question: Is  $L(M)$  recursive?

– Answer: Don't know, we don't have enough information.

– Question: Is  $L(M)$  in r.e – r?

– Answer: Don't know, we don't have enough information.

- **Let  $M$  be a TM that halts on all inputs:**

- Question: Is  $L(M)$  recursively enumerable? – Answer: Yes! By definition it is!
- Question: Is  $L(M)$  recursive?
- Answer: Yes! By definition it is!
- Question: Is  $L(M)$  in r.e. – r?
- Answer: No! It can't be. Since  $M$  always halts,  $L(M)$  is recursive.
- **Let  $M$  be a TM.**
  - As noted previously,  $L(M)$  is recursively enumerable, but may or may not be recursive.
  - Question: Suppose, we know  $L(M)$  is recursive. Does that mean  $M$  always halts?
  - Answer: Not necessarily. However, some TM  $M'$  must exist such that  $L(M') = L(M)$  and  $M'$  always halts.
  - Question: Suppose that  $L(M)$  is in r.e. – r. Does  $M$  always halt?
  - Answer: No! If it did then  $L(M)$  would be recursive and therefore not in r.e. – r.
- **Let  $M$  be a TM, and suppose that  $M$  loops forever on some string  $x$ .**
  - Question: Is  $L(M)$  recursively enumerable?
  - Answer: Yes! By definition it is. But, obviously  $x$  is not in  $L(M)$ .
  - Question: Is  $L(M)$  recursive?



– Answer: Don't know. Although  $M$  doesn't always halt, some other TM  $M'$  may exist such that  $L(M') = L(M)$  and  $M'$  always halts.

– Question: Is  $L(M)$  in r.e. – r?

– Answer: Don't know.

May be another  $M'$  will halt on  $x$ , and on all strings! May be no TM for this  $L(M)$  does halt on all strings! We just do not know!

## Modifications of the Basic TM Model

- **Other (Extended) TM Models:**

- One-way infinite tapes
- Multiple tapes and tape heads
- Non-Deterministic TMs
- Multi-Dimensional TMs (n-dimensional tape)
- Multi-Heads
- Multiple tracks

*All of these extensions are equivalent to the basic DTM model*

## *Closure Properties for Recursive and Recursively Enumerable Languages*

- **TMs model General Purpose (GP) Computers:**

- If a TM can do it, so can a GP computer
- If a GP computer can do it, then so can a TM

*If you want to know if a TM can do  $X$ , then some equivalent question are:*

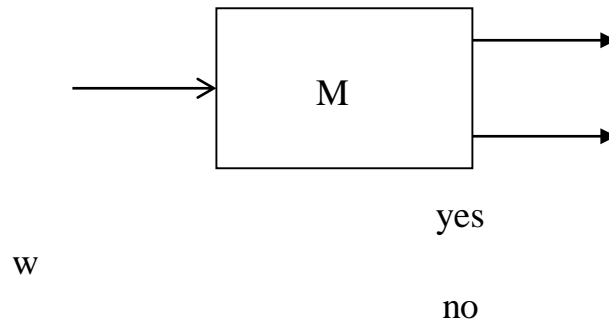
- *Can a general purpose computer do  $X$ ?*
- *Can a C/C++/Java/etc. program be written to do  $X$ ?*

*For example, is a language  $L$  recursive?*

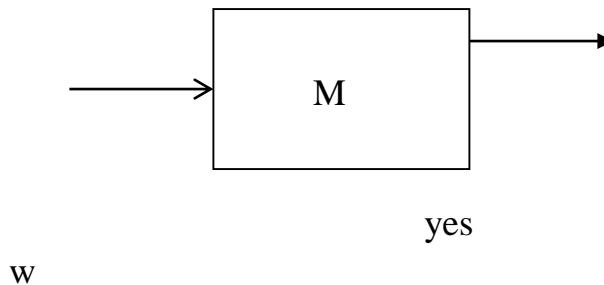
- *Can a C/C++/Java/etc. program be written that always halts and accepts  $L$ ?*

- **TM Block Diagrams:**

- If  $L$  is a recursive language, then a TM  $M$  that accepts  $L$  and always halts can be pictorially represented by a “chip” or “box” that has one input and two outputs.

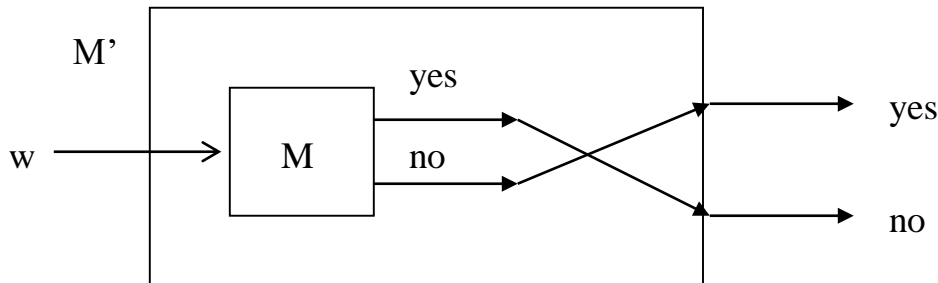


- If  $L$  is a recursively enumerable language, then a TM  $M$  that accepts  $L$  can be pictorially represented by a “box” that has one output.



- Conceivably,  $M$  could be provided with an output for “no,” but this output cannot be counted on. Consequently, we simply ignore it.

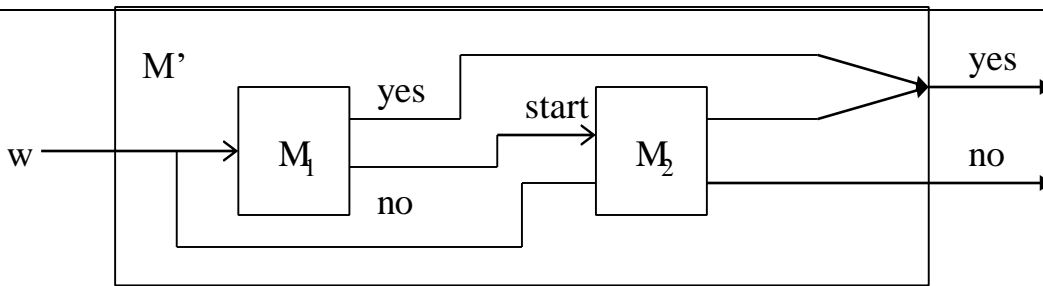
- **Theorem 1:** The recursive languages are closed with respect to complementation, i.e., if  $L$  is a recursive language, then so is  $L^c$ .
- **Proof:** Let  $M$  be a TM such that  $L = L(M)$  and  $M$  always halts. Construct TM  $M'$  as follows:



- **Note That:**
  - $M'$  accepts iff  $M$  does not
  - $M'$  always halts since  $M$  always halts

From this it follows that the complement of  $L$  is recursive. •

- **Question:** How is the construction achieved? Do we simply complement the final states in the TM? No! A string in  $L$  could end up in the complement of  $L$ .
  - Suppose  $q_5$  is an accepting state in  $M$ , but  $q_0$  is not.
  - If we simply complemented the final and non-final states, then  $q_0$  would be an accepting state in  $M'$  but  $q_5$  would not.
  - Since  $q_0$  is an accepting state, by definition all strings are accepted by  $M'$
- **Theorem 2:** The recursive languages are closed with respect to union, i.e., if  $L_1$  and  $L_2$  are recursive languages, then so is  $L_1 \cup L_2$ .
- **Proof:** Let  $M_1$  and  $M_2$  be TMs such that  $L_1 = L(M_1)$  and  $L_2 = L(M_2)$  and  $M_1$  and  $M_2$  always halts. Construct TM  $M'$  as follows:



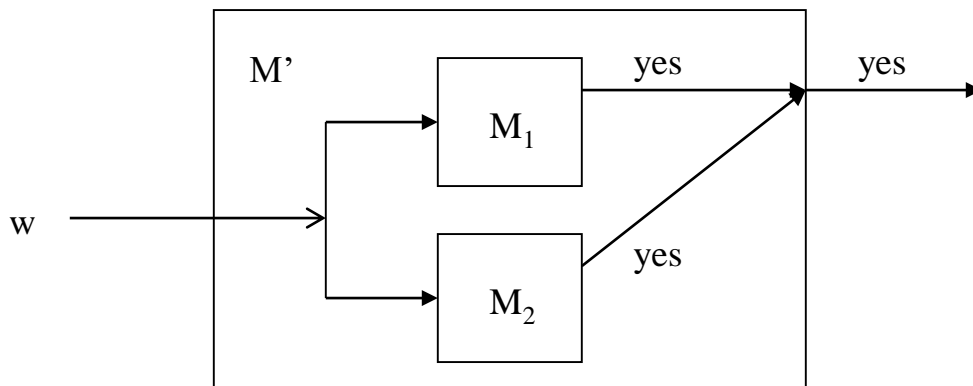
- **Note That:**

- $L(M') = L(M_1) \cup L(M_2)$
- $L(M')$  is a subset of  $L(M_1) \cup L(M_2)$
- $L(M_1) \cup L(M_2)$  is a subset of  $L(M')$ 
  - $M'$  always halts since  $M_1$  and  $M_2$  always halt

It follows from this that

is recursive.  $L_3 \sqsubseteq L_1 \sqcup L_2$  •

- **Theorem 3:** The *recursive enumerable languages* are closed with respect to union, i.e., if  $L_1$  and  $L_2$  are recursively enumerable languages, then so is  $L_3 \sqsubseteq L_1 \sqcup L_2$
- **Proof:** Let  $M_1$  and  $M_2$  be TMs such that  $L_1 = L(M_1)$  and  $L_2 = L(M_2)$ . Construct  $M'$  as follows:



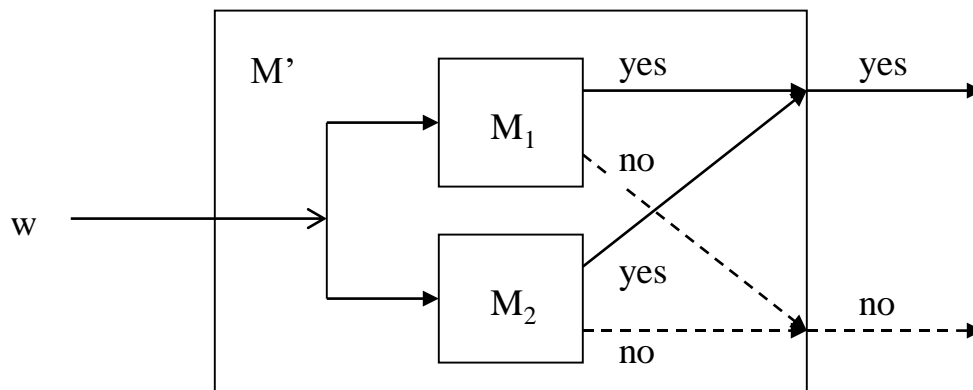
- **Note That:**
  - $L(M') = L(M_1) \cup L(M_2)$
- $L(M')$  is a subset of  $L(M_1) \cup L(M_2)$  •  $L(M_1) \cup L(M_2)$  is a subset of  $L(M')$ 
  - $M'$  halts and accepts iff  $M_1$  or  $M_2$  halts and accepts

It follows from this that

is recursively enumerable.  $L_3 \sqsubseteq L_1 \sqsubseteq L_2$

•

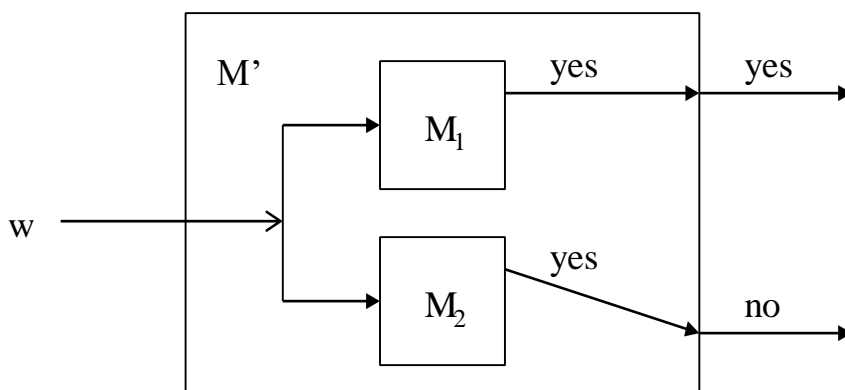
- **Question:** How do you run two TMs in parallel?
- Suppose,  $M_1$  and  $M_2$  had outputs for “no” in the previous construction, and these were transferred to the “no” output for  $M'$



- **Question:** What would happen if  $w$  is in  $L(M_1)$  but not in  $L(M_2)$ ?
- **Answer:** You could get two outputs – one “yes” and one “no.”
  - At least  $M_1$  will halt and answer accept,  $M_2$  may or may not halt.
  - As before, for the sake of convenience the “no” output will be ignored.

- **Theorem 4:** If  $L$  and  $\bar{L}$  are both recursively enumerable then  $L = \bar{L}$  (and therefore  $L$  is recursive).

- **Proof:** Let  $M_1$  and  $M_2$  be TMs such that  $L = L(M_1)$  and  $\bar{L} = L(M_2)$ . Construct  $M'$  as follows:



- **Note That:**

- $L(M') = L$
- $L(M')$  is a subset of  $L$
- $L$  is a subset of  $L(M')$ 
  - $M'$  is TM for  $L$
  - $M'$  always halts since either  $M_1$  or  $M_2$  halts for any given string
  - $M'$  shows that  $L$  is recursive

It follows from this that  $L$  (and therefore its complement) is recursive.

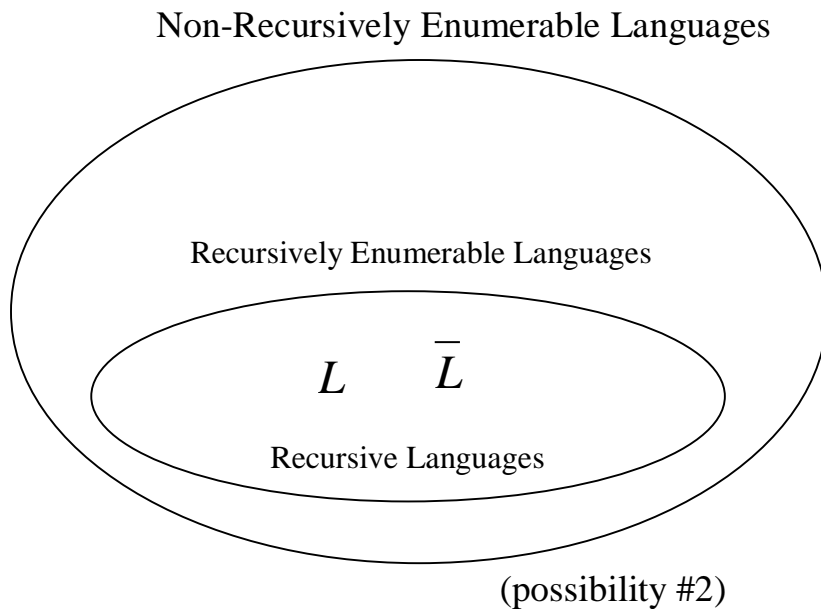
So,  $\bar{L}$  is also recursive (we proved it before).  $L = \bar{L}$

- **Corollary of Thm 4:** Let  $L$  be a subset of  $\Sigma^*$ . Then one of the following must be true:

- Both  $L$  and  $\bar{L}$  are recursive.
- One of  $L$  and  $\bar{L}$  is recursively enumerable but not recursive, and the other is not recursively enumerable, or

- Neither  $L$  nor  $\bar{L}$  is recursively enumerable
- *In other words, it is impossible to have both  $L$  and  $\bar{L}$  r.e. but not recursive*

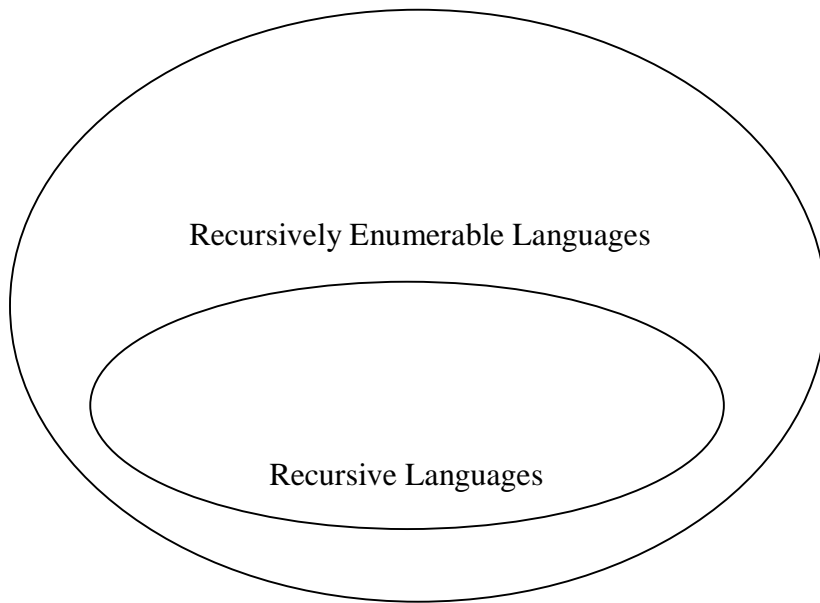
(possibility #1)



Non-Recursively Enumerable Languages

$L$

$\bar{L}$



Recursive Languages

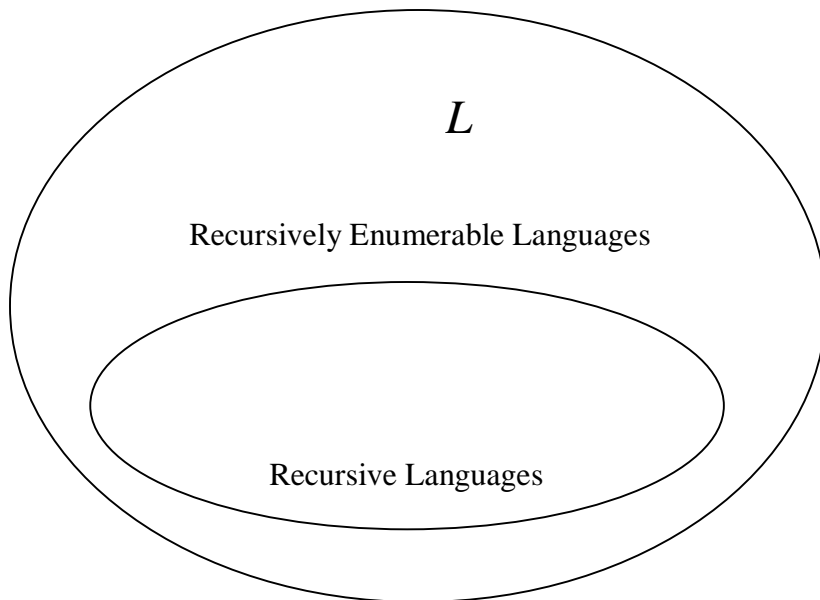
Recursively Enumerable Languages

(possibility #3)

Non-Recursively Enumerable Languages

$L$

$\bar{L}$



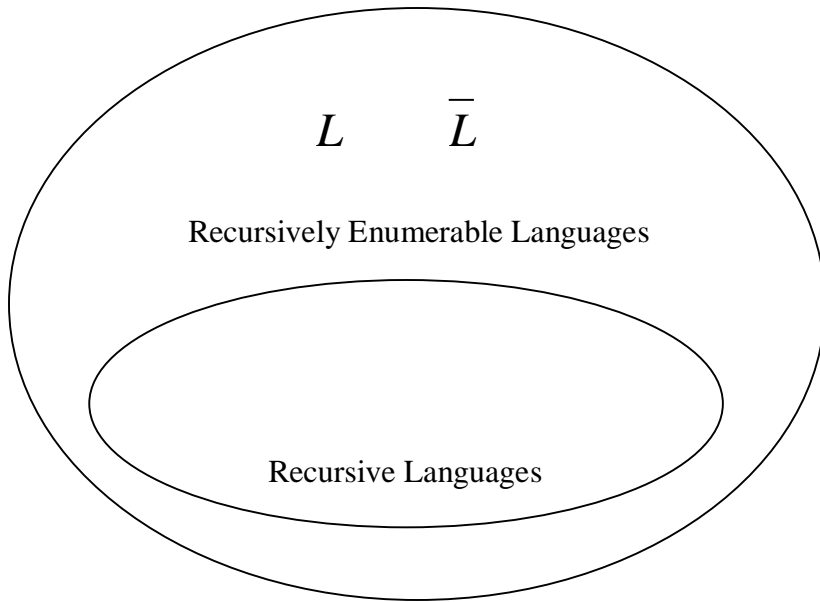
Recursive Languages

Recursively Enumerable Languages



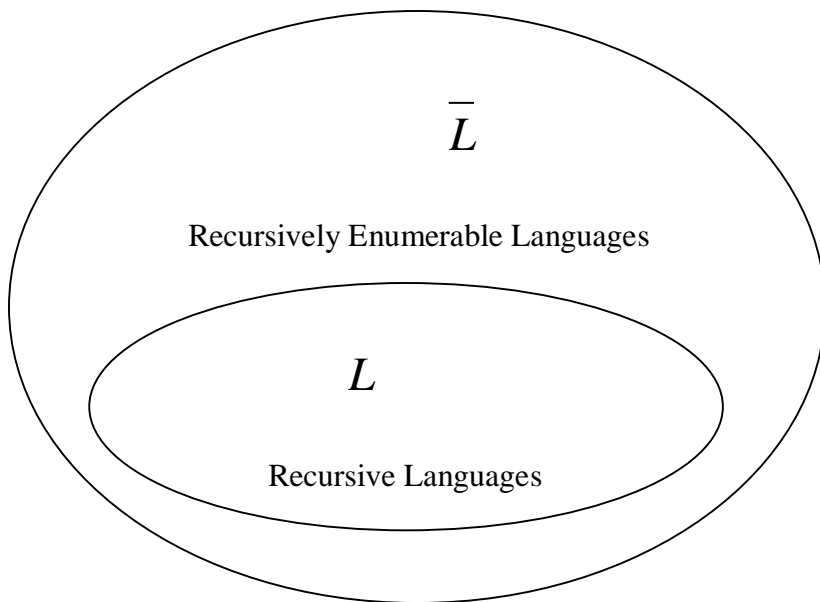
#1)

Non-Recursively Enumerable Languages



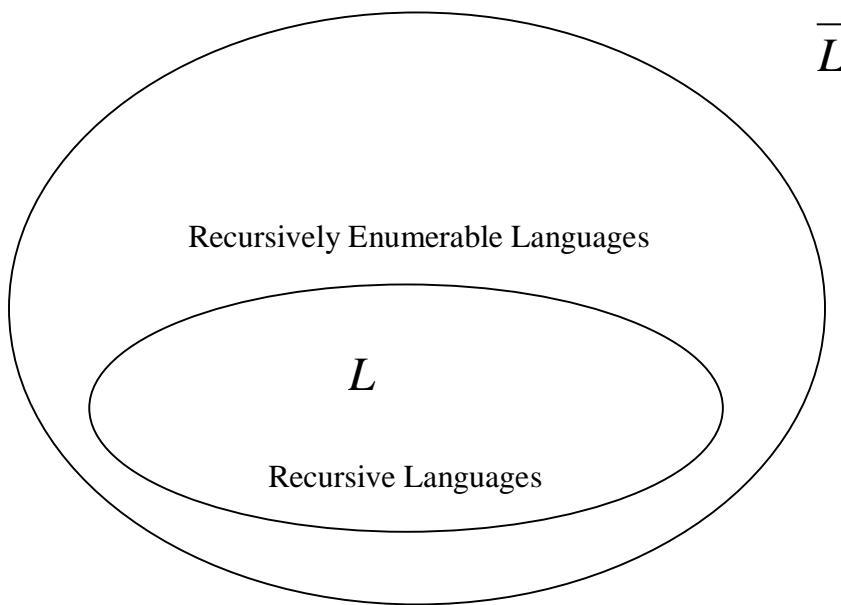
#2)

Non-Recursively Enumerable Languages



#3)

## Non-Recursively Enumerable Languages



- **Note:** This gives/identifies three approaches to show that a language is not recursive.
  - Show that the language's complement is not recursive, in one of the two ways:
    - Show that the language's complement is recursively enumerable but not recursive
    - Show that the language's complement is not even recursively enumerable

### *The Halting Problem - Background*

- **Definition:** A decision problem is a problem having a yes/no answer (that one presumably wants to solve with a computer). Typically, there is a list of parameters on which the problem is based.
  - Given a list of numbers, is that list sorted?
  - Given a number  $x$ , is  $x$  even?
  - Given a C program, does that C program contain any syntax errors?
  - Given a TM (or C program), does that TM contain an infinite loop?

From a practical perspective, many decision problems do not seem all that interesting. However, from a theoretical perspective they are for the following two reasons:

- Decision problems are more convenient/easier to work with when proving complexity results.

- Non-decision *counter-parts* can always be created & are typically at least as difficult to solve.

- **Notes:**

- The following terms and phrases are analogous:

Algorithm	- A halting TM program
Decision Problem	- A language ( <i>will show shortly</i> )
(un)Decidable	- (non)Recursive

### *Statement of the Halting Problem*

- **Practical Form: (P1)**

Input: Program P and input I.

Question: Does P terminate on input I?

- **Theoretical Form: (P2)** Input: Turing machine M with input alphabet  $\Sigma$  and string w in  $\Sigma^*$ . Question: Does M halt on w?

- **A Related Problem We Will Consider First: (P3)** Input: Turing machine M with input alphabet  $\Sigma$  and one final state, and string w in  $\Sigma^*$ .

Question: Is w in L(M)?

- **Analogy:**

Input: DFA M with input alphabet  $\Sigma$  and string w in  $\Sigma^*$ .

Question: Is w in L(M)?

Is this problem (*regular language*) decidable? Yes! DFA always accepts or rejects.

- **Over-All Approach:**

- We will show that a language  $L_d$  is not recursively enumerable
- From this it will follow that  $\overline{L_d}$  is not recursive
- Using this we will show that a language  $L_u$  is not recursive – From this it will follow that the halting problem is undecidable.

- **As We Will See:**

- P3 will correspond to the language  $L_u$
- Proving P3 (un)decidable is equivalent to proving  $L_u$  (non)recursive

*Converting the Problem to a Language*

- Let  $M = (Q, \Sigma, \Gamma, \delta, q_1, B, \{q_n\})$  be a TM, where

$Q = \{q_1, q_2, \dots, q_n\}$ , order the states from 1 through  $n$

$\Sigma = \{x_1, x_2\} = \{0, 1\}$   $\Gamma = \{x_1, x_2, x_3\} = \{0, 1, B\}$

- Encode each transition:

$\delta(q_i, x_j) = (q_k, x_l, d_m)$

where  $q_i$  and  $q_k$  are in ordered  $Q$

$x_j$  and  $x_l$  are in  $\Sigma$ , and  $d_m$  is in  $\{L, R\} = \{d_1, d_2\}$

as:

$0^i 10^j 10^k 10^l 10^m$  where the number of 0's indicate the corresponding id, and single 1 acts as a barrier

- The TM  $M$  can then be encoded as:

$111\text{code}_1 11\text{code}_2 11\text{code}_3 11 \dots 11\text{code}_r 111$

where each  $\text{code}_i$  is one transitions' encoding, and 11's are barriers between transitions from the table row-major. Let this encoding of  $M$  be denoted by  $\langle M \rangle$ .

- Less Formally:

- Every state, tape symbol, and movement symbol is encoded as a sequence of 0's:

$q_1, 0 \ q_2, 00$

$q_3: 000$

0      0

1      00

B      000

L      0

R      00

- Note that 1's are not used to represent the above, since 1 is used as a special separator symbol.
- Example:

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

Is encoded as:

00100100010100

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

$q_2$

$q_3$

*What is the  $L(M)$ ?*

Coding for the above table:

111010101010011101001010010011101000100100010110010100010100111

Are the followings correct encoding of a TM?

01100001110001

111111

- **Definition:**

$$L_t = \{x \mid x \text{ is in } \{0, 1\}^* \text{ and } x \text{ encodes a TM}\}$$

- Question: Is  $L_t$  recursive?
- Answer: Yes. [Check only for format, i.e. the order and number of 0's and 1's, syntax checking]
- Question: Is  $L_t$  decidable:
- Answer: Yes (same question).

### *The Universal Language*

- Define the language  $L_u$  as follows:

$$L_u = \{x \mid x \text{ is in } \{0, 1\}^* \text{ and } x = \langle M, w \rangle \text{ where } M \text{ is a TM encoding and } w \text{ is in } L(M)\}$$

- Let  $x$  be in  $\{0, 1\}^*$ . Then either:

1.  $x$  doesn't have a TM prefix, in which case  $x$  is **not** in  $L_u$

2.  $x$  has a TM prefix, i.e.,  $x = \langle M, w \rangle$  and either:

a)  $w$  is not in  $L(M)$ , in which case  $x$  is **not** in  $L_u$

b)  $w$  is in  $L(M)$ , in which case  $x$  is in  $L_u$

- **Recall:**

	0	1	B
q <sub>1</sub>	(q <sub>1</sub> , 0, R)	(q <sub>1</sub> , 1, R)	(q <sub>2</sub> , B, L)
q <sub>2</sub>	(q <sub>3</sub> , 0, R)	-	-
q <sub>3</sub>	-	-	-

- **Which of the following are in  $L_u$ ?**

1110101010100110100101001001101000100100010110010100010100111

111010101010011010010100100110100010010001011001010001010011101110

111010101010011010010100100110100010010001011001010001010011100110111

01100001110001

111111

- **Compare P3 and  $L_u$ :**

(P3):

Input: Turing machine  $M$  with input alphabet  $\Sigma$  and one final state, and string  $w$  in  $\Sigma^*$ .

Question: Is  $w$  in  $L(M)$ ?

$L_u = \{x \mid x \text{ is in } \{0, 1\}^* \text{ and } x = \langle M, w \rangle \text{ where } M \text{ is a TM encoding and } w \text{ is in } L(M)\}$

- Universal TM ( $UTM$ ) is the machine for  $L_u$ 
    - presuming it is r.e.! *Can you write a program to accept strings in  $L_u$ ?*
  - **Notes:**
    - $L_u$  is P3 expressed as a language
    - *Asking if  $L_u$  is recursive is the same as asking if P3 is decidable.*
  - *Can you write a Halting program for accept/reject of strings in  $\Sigma^*$  ?*
    - We will show that  $L_u$  is *not recursive*, and from this it will follow that P3 is *undecidable*.
    - From this we can further show that the *Halting problem is un-decidable*.
- => A general concept: *a decision problem  $\equiv$  a formal language*
- Define another language  $L_d$  as follows:
  - $[L_d\_bar = \{\text{self accepting TM encodings}\}, \quad \text{everything else is } L_d]$

$L_d = \{x \mid x \text{ is in } \{0, 1\}^* \text{ and (a) either } x \text{ is **not** a TM,$

(b) or  $x$  is a TM, call it  $M$ , and  $x$  is **not** in  $L(M)$  (1)

- Note, there is only one string  $x$
- And, the question really is the complement of “does a TM accept its own encoding?” ( $L_d$ -bar’s complement)

- Let  $x$  be in  $\{0, 1\}^*$ . Then either:

1.  $x$  is **not** a TM, in which case  $x$  is in  $L_d$   
 2.  $x$  is a TM, call it  $M$ , and either:

- a)  $x$  is **not** in  $L(M)$ , in which case  $x$  is in  $L_d$
- b)  $x$  is in  $L(M)$ , in which case  $x$  is **not** in  $L_d$

- **Recall:**

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

- **Which of the following are in  $L_d$ ?**

11101010101001101001010010011010001000100010110010100010100111

01100001110001

*Change above machine to accept strings ending with 1: the encoding will not be in  $L_d$*



- **Lemma:**  $L_d$  is not recursively enumerable. [No TM for  $L_d$ !!!]

- **Proof:** (by contradiction)

Suppose that  $L_d$  is recursively enumerable. In other words, there exists a TM  $M$  such that:

$$L_d = L(M) \quad (2)$$

Now suppose that  $w$  is a string encoding of  $M$ . (3)

Case 1)  $w$  is in  $L_d$  (4)

By definition of  $L_d$  given in (1), either  $w$  does not encode a TM, or  $w$  does encode a TM, call it  $M$ , and  $w$  is not in  $L(M)$ . But we know that  $w$  encodes a TM (3: that's where it came from). Therefore:

$$w \text{ is not in } L(M) \quad (5)$$

But then (2) and (5) imply that  $w$  is not in  $L_d$  contradicting (4).

Case 2)  $w$  is not in  $L_d$  (6)

By definition of  $L_d$  given in (1),  $w$  encodes a TM, call it  $M$ , and:

$$w \text{ is in } L(M) \quad (7)$$

But then (2) and (7) imply that  $w$  is in  $L_d$  contradicting (6).

Since both case 1) and case 2) lead to a contradiction, no TM  $M$  can exist such that  $L_d = L(M)$ . Therefore  $L_d$  is not recursively enumerable. •

- **Note:**

$$\overline{L_d} = \{x \mid x \text{ is in } \{0, 1\}^*, x \text{ encodes a TM, call it } M, \text{ and } x \text{ is in } L(M)\} \text{ ---}$$

**Corollary:**  $L_d$  is not recursive.

- **Proof:** If  $\overline{L_d}$  were recursive, then  $L_d$  would be recursive, and therefore recursively enumerable, a contradiction. •

- **Theorem:**  $L_u$  is not recursive.

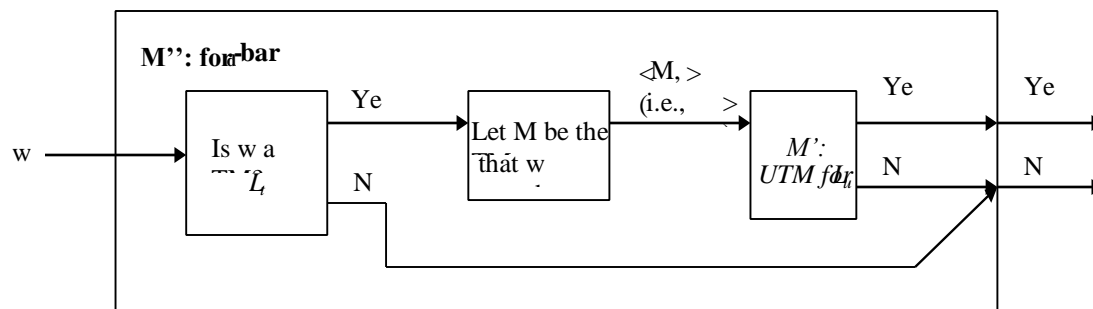
- **Proof:** (by contradiction)

Suppose that  $L_u$  is recursive. Recall that:

$$L_u = \{x \mid x \text{ is in } \{0, 1\}^* \text{ and } x = \langle M, w \rangle \text{ where } M \text{ is a TM encoding and } w \text{ is in } L(M)\}$$

Suppose that  $L_u = L(M')$  where  $M'$  is a TM that always halts. Construct an algorithm (i.e., a TM that

always halts) for  $\overline{L_d}$  as follows:



Suppose that  $M'$  always halts and  $L_u = L(M')$ . It follows that:

- $M''$  always halts

- $L(M'') = \overline{L_d}$

$\overline{L_d}$  would therefore be recursive, a contradiction. •

**$L_u$  is recursively enumerable**

(you may ignore this slide, for now)

Input the string

Decode the TM prefix, if it doesn't have one then the string is not in  $L_u$

Otherwise, run/simulate the encoded TM on the suffix

If it terminates and accepts then the original string is in  $L_u$ .

If a given string is in  $L_u$ , then the above algorithm will correctly determine that, halt and say yes.

If the given string is not in  $L_u$ , then there are three cases:

- 1) the string doesn't have a TM as a prefix. In this case the above algo correctly detects this fact, and reports the string is not in  $L_u$ .
- 2) the string has a TM prefix, and the TM halts and rejects on the suffix. In this case the above algo correctly reports the string is not in  $L_u$ .
- 3) the string has a TM prefix, but it goes into an infinite loop on the suffix. In this case the above algo also goes into an infinite loop, but that's ok since the string as a whole is not in  $L_u$  anyway, and we are just trying to show there exists a TM for only accepting strings in  $L_u$ .

From this proof note that if the prefix TM is a DFA or PDA, then our machine will also halt in the 3<sup>rd</sup> case above, no matter what the suffix is.

-- due to Dr. Bernhard (edited by me)

- **The over-all logic of the proof is as follows:**

1. If  $L_u$  were recursive, then so will be  $L_d$
2.  $L_d$  is not recursive, because  $L_d$  is not r.e.
3. It follows that  $L_u$  is not recursive.

The second point was established by the corollary.

The first point was established by the theorem on a preceding slide.

This type of proof is commonly referred to as a *reduction*. Specifically, the problem of recognizing  $L_d$  was *reduced* to the problem of recognizing  $L_u$

- **Define another language  $L_h$ :**

$$L_h = \{x \mid x \text{ is in } \{0, 1\}^* \text{ and } x = \langle M, w \rangle \text{ where } M \text{ is a TM encoding and } M \text{ halts on } w\}$$

Note that  $L_h$  is P2 expressed as a language:

(P2):

Input: Turing machine  $M$  with input alphabet  $\Sigma$  and string  $w$  in  $\Sigma^*$ .

Question: Does  $M$  halt on  $w$ ?

- **Theorem:**  $L_h$  is not recursive.

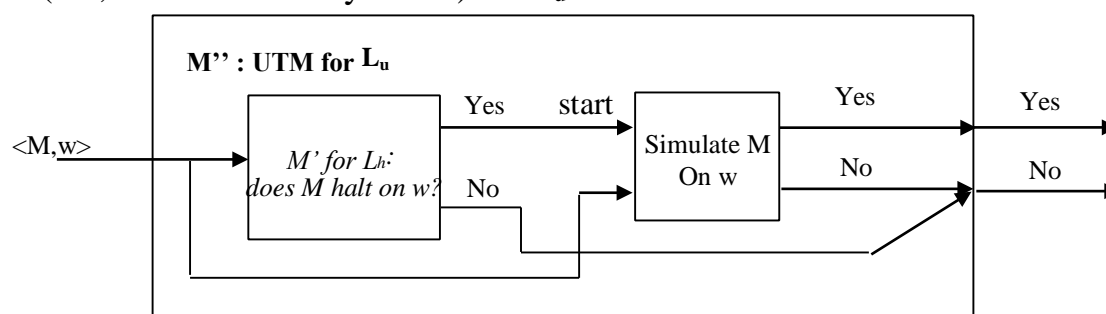
- **Proof:** (by contradiction)

Suppose that  $L_h$  is recursive. Recall that:

$L_h = \{x \mid x \text{ is in } \{0, 1\}^* \text{ and } x = \langle M, w \rangle \text{ where } M \text{ is a TM encoding and } M \text{ halts on } w\}$  and

$L_u = \{x \mid x \text{ is in } \{0, 1\}^* \text{ and } x = \langle M, w \rangle \text{ where } M \text{ is a TM encoding and } w \text{ is in } L(M)\}$

Suppose that  $L_h = L(M')$  where  $M'$  is a TM that always halts. Construct an algorithm (i.e., a TM that always halts) for  $L_u$  as follows:



Suppose that  $M'$  always halts and  $L_h = L(M')$ . It follows that:

- $M''$  always halts
- $L(M'') = L_u$

$L_u$  would therefore be recursive, a contradiction. •

- **The over-all logic of the proof is as follows:**

1. If  $L_h$  is recursive, then so is  $L_u$
2.  $L_u$  is not recursive
3. It follows that  $L_h$  is not recursive.

The second point was established previously.

The first point was established by the theorem on the preceding slide.

This proof is also a reduction. Specifically, the problem of recognizing  $L_u$  was *reduced* to the problem of recognizing  $L_h$ .

*[ $L_u$  and  $L_h$  both are recursively enumerable: for proof see Dr. Shoaft!]*

Examples of non-halting program:

- **Define another language  $L_q$ :**

$L_q = \{x \mid x \text{ is in } \{0, 1\}^*, x \text{ encodes a TM } M, \text{ and } M \text{ does **not** contain an infinite loop}\}$

Or equivalently:

$L_q = \{x \mid x \text{ is in } \{0, 1\}^*, x \text{ encodes a TM } M, \text{ and there exists **no** string } w \text{ in } \{0, 1\}^* \text{ such that } M \text{ does **not** terminate on } w\}$

Note that:

$\overline{L_q} = \{x \mid x \text{ is in } \{0, 1\}^*, \text{ and either } x \text{ does **not** encode a TM, or it does encode a TM, call it } M, \text{ and there exists a string } w \text{ in } \{0, 1\}^* \text{ such that } M \text{ does **not** terminate on } w\}$

Note that the above languages correspond to the following problem:

(P0):

Input: Program P.

Question: Does P contain an infinite loop?

*Using the techniques discussed, what can we prove about  $L_q$  or its' complement?*

- **More examples of non-recursive languages:**

$L_{ne} = \{x \mid x \text{ is a TM } M \text{ and } L(M) \text{ is not empty}\}$  is r.e. but not recursive.

$L_e = \{x \mid x \text{ is a TM } M \text{ and } L(M) \text{ is empty}\}$  is not r.e.

$L_r = \{x \mid x \text{ is a TM } M \text{ and } L(M) \text{ is recursive}\}$  is not r.e.

Note that  $L_r$  is not the same as  $L_h = \{x \mid x \text{ is a TM } M \text{ that always halts}\}$   
but  $L_h$  is in  $L_r$ .

$L_{nr} = \{x \mid x \text{ is a TM } M \text{ and } L(M) \text{ is not recursive}\}$  is not r.e.

**Ignore this slide**

- **Lemma:**  $L_d$  is not recursively enumerable: [No TM for  $L_d$ !!!]
- **Proof:** (by contradiction)  
Suppose that  $L_d$  were recursively enumerable. In other words, that there existed a TM  $M$  such that:

$$L_d = L(M) \quad (2)$$

Now suppose that  $w_j$  is a string encoding of  $M$ . (3)

Case 1)  **$w_j$  is in  $L_d$**  (4)

By definition of  $L_d$  given in (1), either  $w_j$  does not encode a TM, or  $w_j$  does encode a TM, call it  $M$ , and  $w_j$  is not in  $L(M)$ . But we know that  $w_j$  encodes a TM (3: that's where it came from). Therefore:

$$w_j \text{ is not in } L(M) \quad (5)$$

But then (2) and (5) imply that  **$w_j$  is not in  $L_d$**  contradicting (4).

Case 2)  **$w_j$  is not in  $L_d$**  (6)

By definition of  $L_d$  given in (1),  $w_j$  encodes a TM, call it  $M$ , and:

$$w_j \text{ is in } L(M) \quad (7)$$

But then (2) and (7) imply that  **$w_j$  is in  $L_d$**  contradicting (6).

Since both case 1) and case 2) lead to a contradiction, no TM  $M$  can exist such that  $L_d = L(M)$ . Therefore  $L_d$  is not recursively enumerable. •

*Roger's TM for balanced parenthesis:*

<b>findPair</b>	(findPair2, "(", R)	-	(final, B, R)
<b>findPair2</b>	(findPair2, "(", R)	(removePair, ")", L)	-
<b>removePair</b>	(fetch, "(", R)	(fetch, ")", R)	(goBack, B, L)
<b>fetch</b>	(retrieve, "(", R)	(retrieve, ")", R)	(retrieve, B, R)
<b>retrieve</b>	(returnOpen, "(", L)	(returnClosed, ")", L)	(returnBlank, B, L)
<b>returnOpen</b>	(writeOpen, "(", L)	(writeOpen, ")", L)	(writeOpen, B, L)
<b>returnClosed</b>	(writeClosed, "(", L)	(writeClosed, ")", L)	(writeClosed, B, L)
<b>returnBlank</b>	(writeBlank "(", L)	(writeBlank, ")", L)	(writeBlank, B, L)
<b>writeOpen</b>	(removePair, "(", R)	(removePair, "(", R)	-
<b>writeClosed</b>	(removePair, ")", R)	(removePair, ")", R)	-
<b>writeBlank</b>	(removePair, B, R)	(removePair, B, R)	-
<b>goBack</b>	-	-	(backAgain, B, L)
<b>backAgain</b>	-	-	(seekFront, B, L)
<b>seekFront</b>	(seekFront, "(", L)	(seekFront, ")", L)	(findPair, B, R)
<b>final*</b>	-	-	-

On 111 111 as a TM encoding