

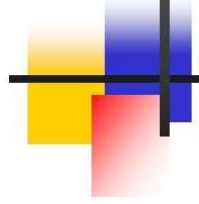


FORMAL LANGUAGES AND AUTOMATA THEORY

UNIT 5



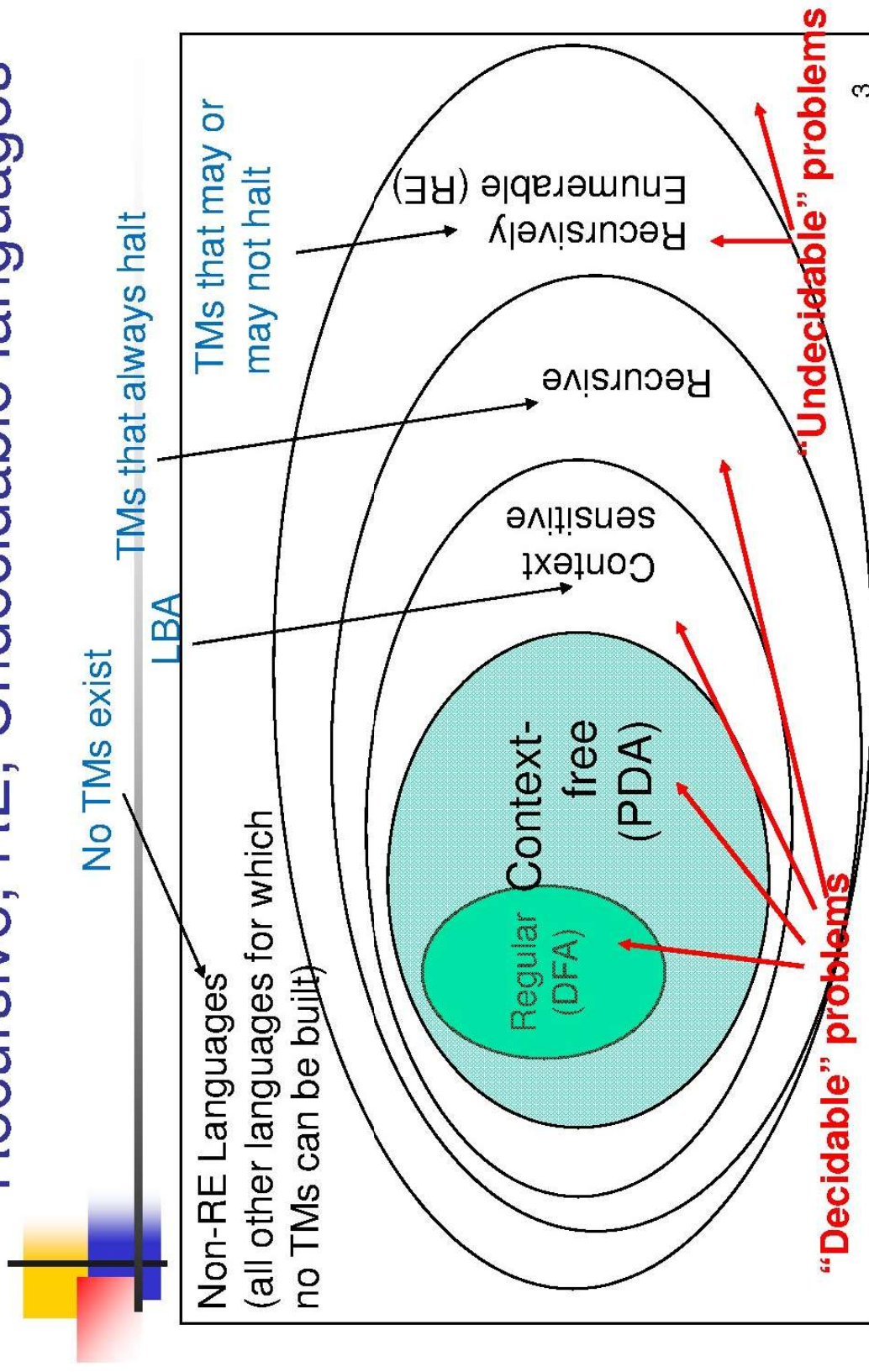
Undecidability



Decidability vs. Undecidability

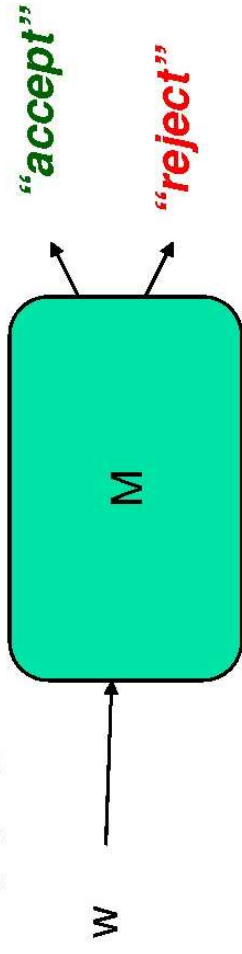
- There are two types of TMs (based on halting):
(*Recursive*)
 - TMs that ***always* halt**, no matter accepting or non-accepting ≡ **DECIDABLE PROBLEMS**
(*Recursively enumerable*)
 - TMs that ***are guaranteed to halt only on acceptance***. If non-accepting, it may or may not halt (i.e., could loop forever).
- **Undecidability:**
 - Undecidable problems are those that are **not** recursive

Recursive, RE, Undecidable languages

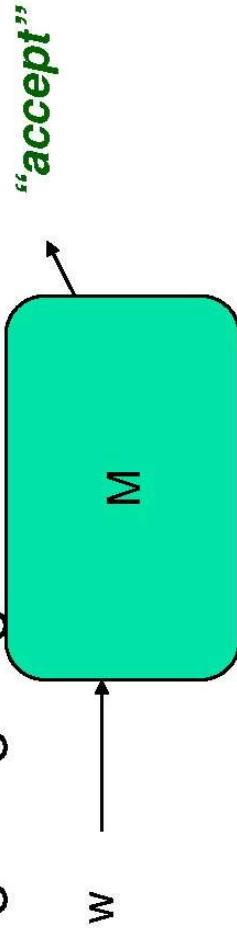


Recursive Languages & Recursively Enumerable (RE) languages

- Any TM for a Recursive language is going to look like this:



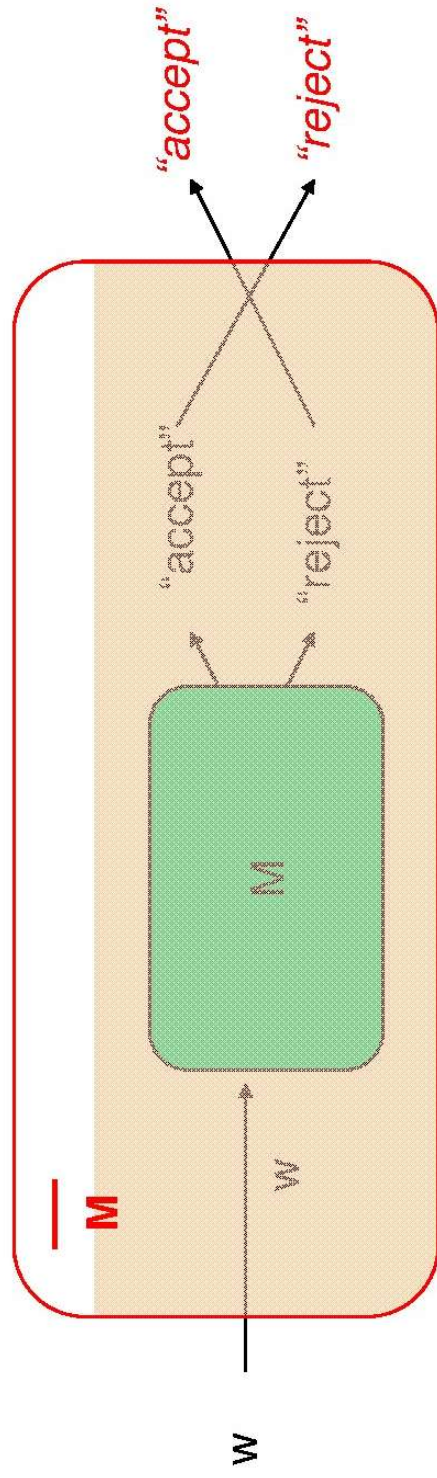
- Any TM for a Recursively Enumerable (RE) language is going to look like this:



- 
- Closure Properties of:
- the Recursive language class, and
 - the Recursively Enumerable language class

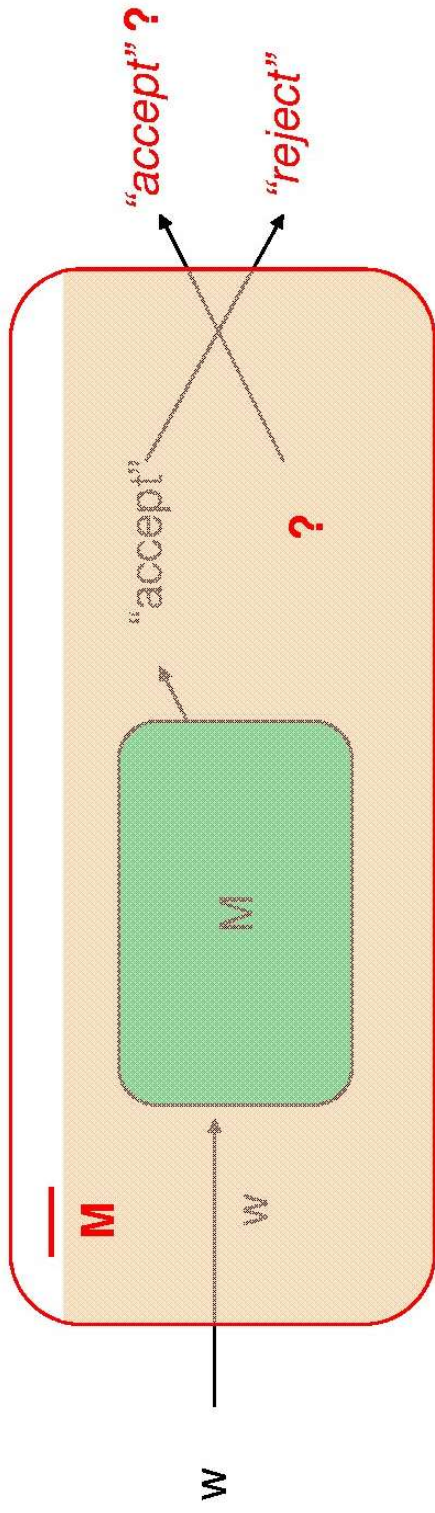
Recursive Languages are closed under complementation

- If L is Recursive, \overline{L} is also Recursive



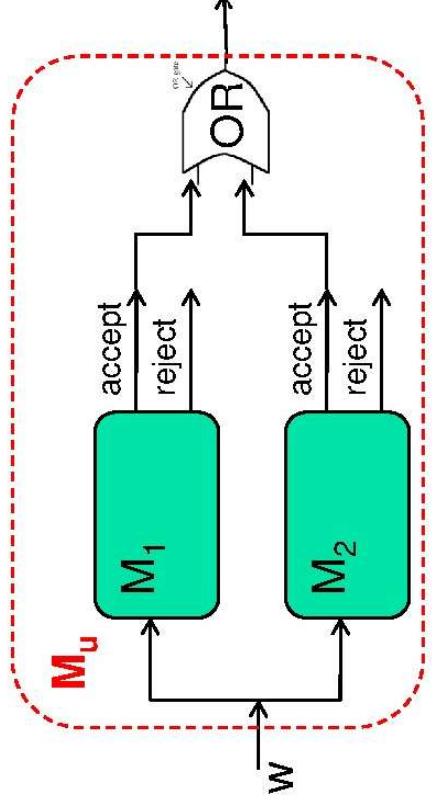
Are Recursively Enumerable Languages closed under complementation? (NO)

- If L is RE, \bar{L} need not be RE



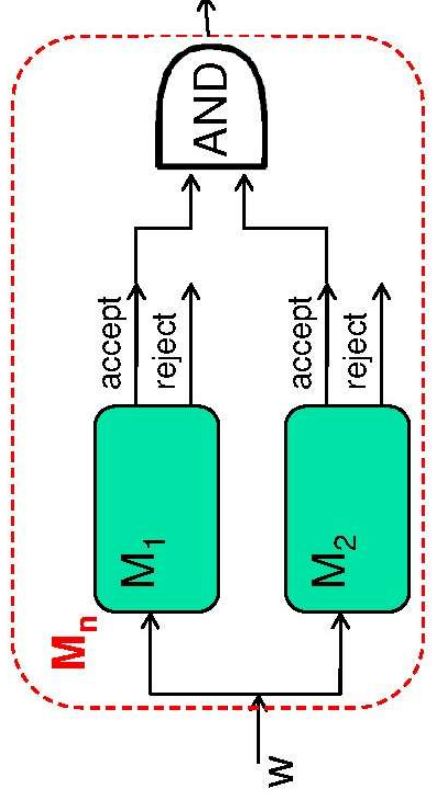
Recursive Languages are closed under Union

- Let $M_u = TM$ for $L_1 \cup L_2$
- M_u construction:
 1. Make 2-tapes and copy input w on both tapes
 2. Simulate M_1 on tape 1
 3. Simulate M_2 on tape 2
 4. If either M_1 or M_2 accepts, then M_u accepts
 5. Otherwise, M_u rejects.



Recursive Languages are closed under Intersection

- Let $M_n = \text{TM}$ for $L_1 \cap L_2$
- M_n construction:
 1. Make 2-tapes and copy input w on both tapes
 2. Simulate M_1 on tape 1
 3. Simulate M_2 on tape 2
 4. If either M_1 AND M_2 accepts, then M_n accepts
 5. Otherwise, M_n rejects.





Other Closure Property Results

- Recursive languages are also closed under:
 - Concatenation
 - Kleene closure (star operator)
 - Homomorphism, and inverse homomorphism
 - RE languages are closed under:
 - Union, intersection, concatenation, Kleene closure
-
- RE languages are *not* closed under:
 - complementation



“Languages” vs. “Problems”

A “language” is a set of strings

Any “problem” can be expressed as a set of all strings that are of the form:

- “<input, output>”

e.g., Problem (a+b) \equiv Language of strings of the form { “a#b, a+b” }

\implies Every problem also corresponds to a language!!

Think of the language for a “problem” \implies a *verifier* for the problem

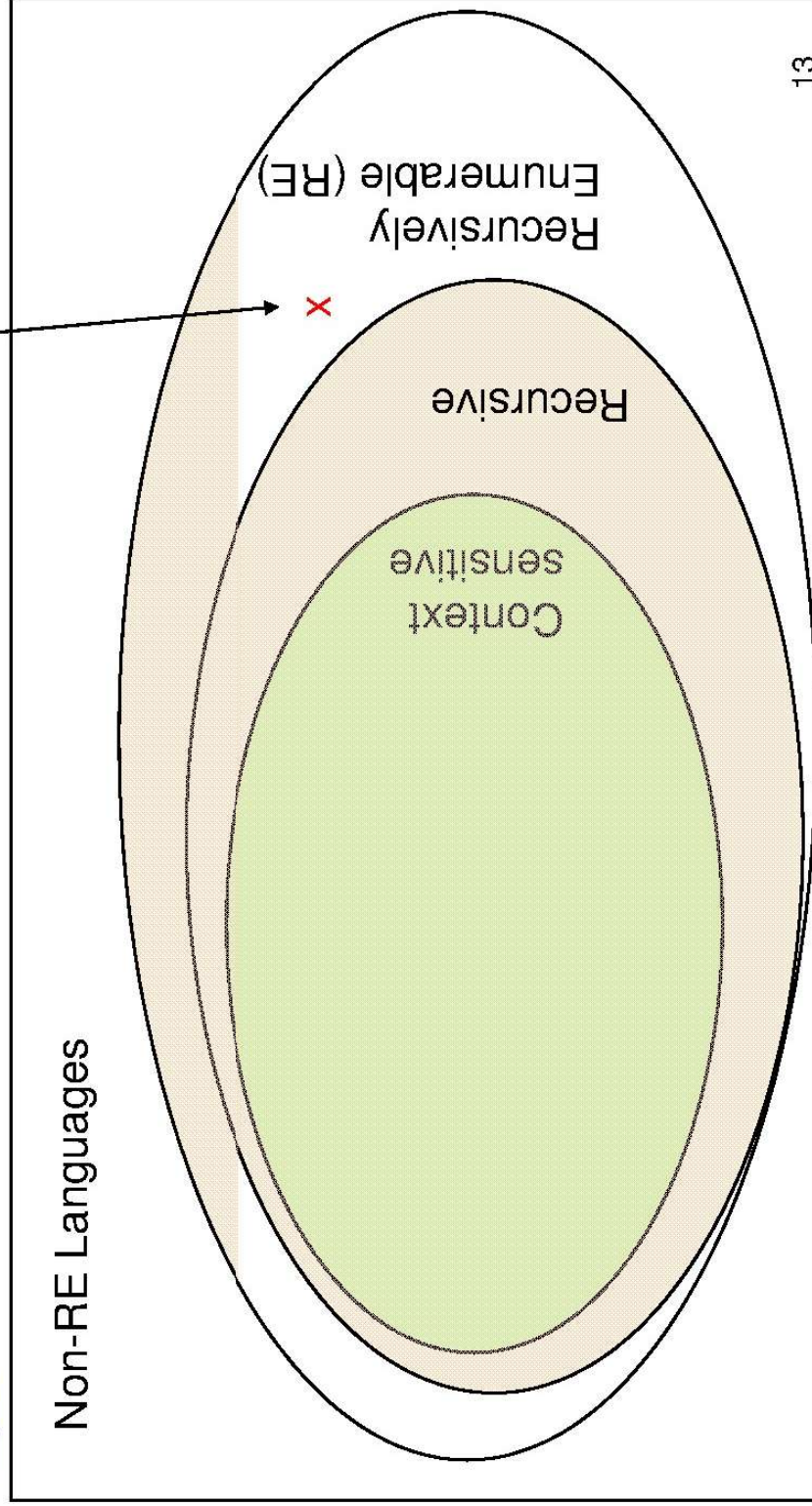


The Halting Problem

An example of a recursive
enumerable problem that is
also undecidable



The Halting Problem

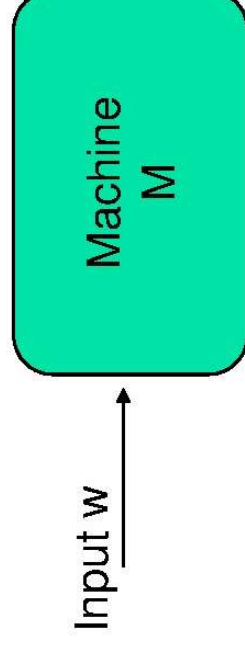




What is the Halting Problem?

Definition of the “halting problem”:

- *Does a given Turing Machine M halt on a given input w ?*



The Universal Turing Machine

A Turing Machine simulator

- Given: TM M & its input w
- Aim: Build another TM called “H”, that will output:
 - “accept” if M accepts w , and
 - “reject” otherwise

- An algorithm for H:
 - Simulate M on w

$$H(\langle M, w \rangle) = \begin{cases} \text{accept,} & \text{if } M \text{ accepts } w \\ \text{reject,} & \text{if } M \text{ does not accept } w \end{cases}$$

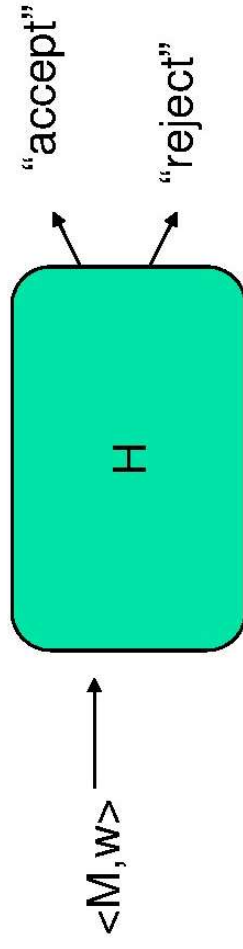
Implies: H is in RE

Question: If M does *not* halt on w , what will happen to H?



A Claim

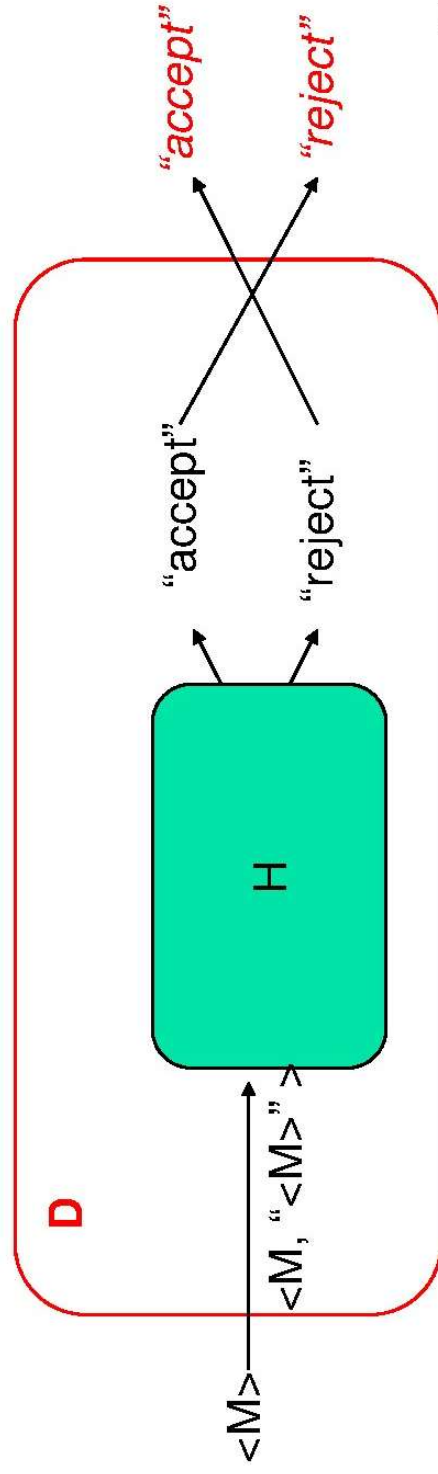
- Claim: No H that is always guaranteed to halt, can exist!
- Proof: (Alan Turing, 1936)
 - By contradiction, let us assume H exists



Therefore, if H exists → D also should exist.
But can such a D exist? (if not, then H also cannot exist)

HP Proof (step 1)

- Let us construct a new TM **D** using H as a subroutine:
 - On input $\langle M \rangle$:
 - Run H on input $\langle M, \langle M \rangle \rangle$; //(i.e., run M on M itself)
 - Output the *opposite* of what H outputs;



HP Proof (step 2)

- The notion of inputting “ $\langle M \rangle$ ” to M itself
 - A program can be input to itself (e.g., a compiler is a program that takes any program as input)

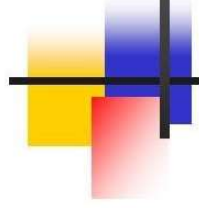
$$D(\langle M \rangle) = \begin{cases} \text{accept,} & \text{if } M \text{ does not accept } \langle M \rangle \\ \text{reject,} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

Now, what happens if D is input to itself?

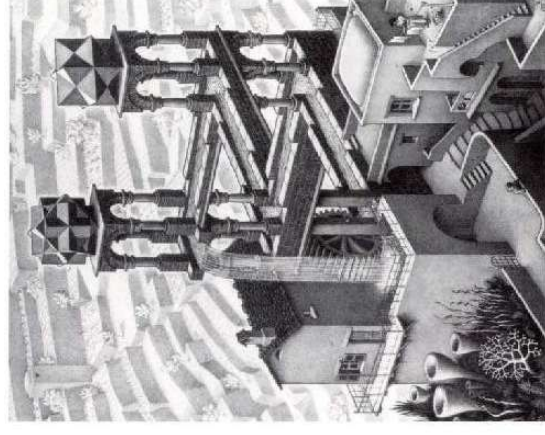
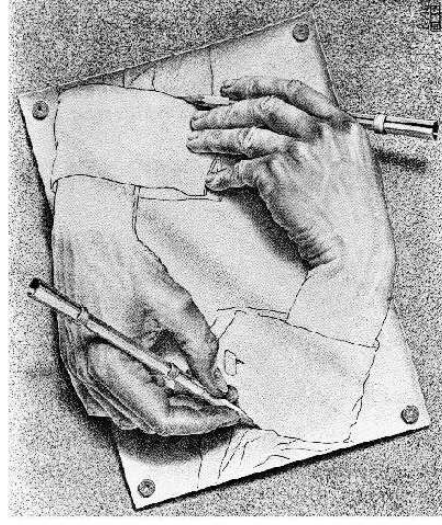
$$D(\langle D \rangle) = \begin{cases} \text{accept,} & \text{if } D \text{ does not accept } \langle D \rangle \\ \text{reject,} & \text{if } D \text{ accepts } \langle D \rangle \end{cases}$$

A contradiction!!! \implies Neither D nor H can exist.

Of Paradoxes & Strange Loops



E.g., Barber's paradox, Achilles & the Tortoise (Zeno's paradox)
MC Escher's paintings



A fun book for further reading:

"Godel, Escher, Bach: An Eternal Golden Braid"
by Douglas Hofstadter (Pulitzer winner, 1980)



The Diagonalization Language

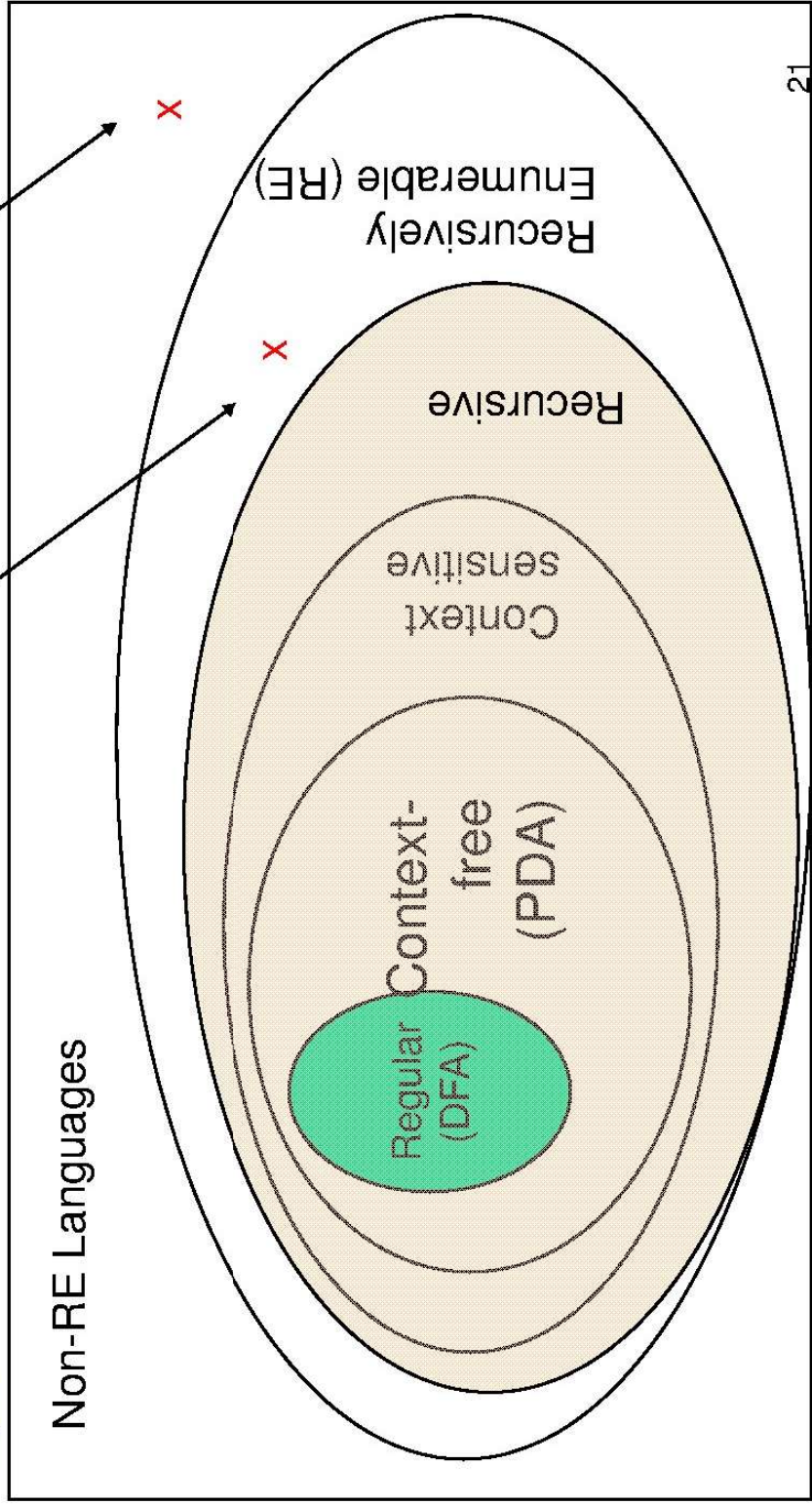
**Example of a language that is
not recursive enumerable**


(i.e, no TMs exist)



The Diagonalization language

The Halting Problem





A Language about TMs & acceptance

- Let L be the language of all strings $\langle M, w \rangle$ s.t.:
 1. M is a TM (coded in binary) with input alphabet also binary
 2. w is a binary string
 3. M accepts input w .



Enumerating all binary strings

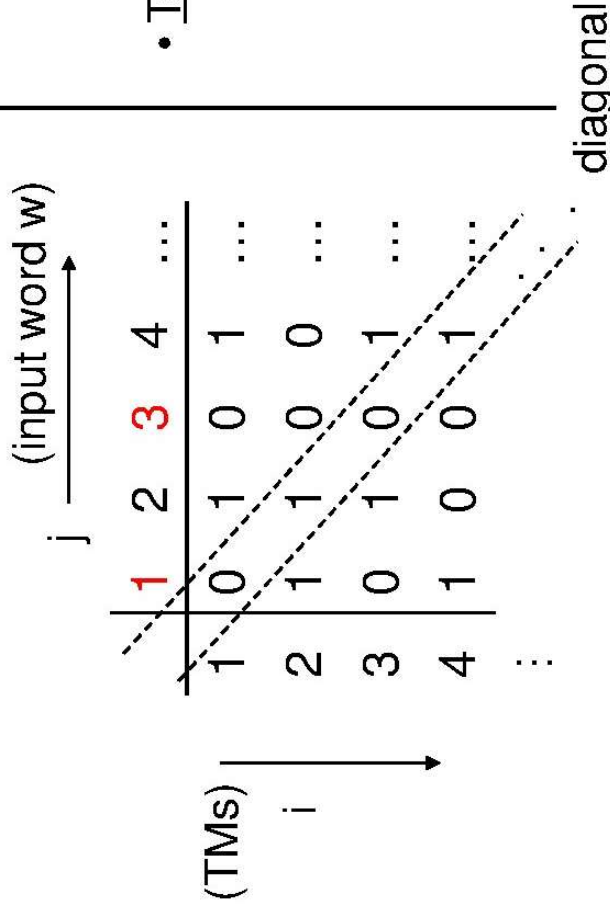
- Let w be a binary string
- Then $1w \equiv i$, where i is some integer
 - E.g., If $w = \epsilon$, then $i = 1$;
 - If $w = 0$, then $i = 2$;
 - If $w = 1$, then $i = 3$; so on...
- If $1w \equiv i$, then call w as the i^{th} word or i^{th} binary string, denoted by w_i .
- \implies **A canonical ordering of all binary strings:**
 - $\{\epsilon, 0, 1, 00, 01, 10, 11, 000, 100, 101, 110, \dots\}$
 - $\{w_1, w_2, w_3, w_4, \dots, w_i, \dots\}$

Any TM M can also be binary-coded

- $M = \{ Q, \{0, 1\}, \Gamma, \delta, q_0, B, F \}$
- Map all states, tape symbols and transitions to integers (\implies binary strings)
- $\delta(q_i, X_j) = (q_k, X_l, D_m)$ will be represented as:
 - $\implies 0^i 1 0^j 1 0^k 1 0^l 1 0^m$
- Result: Each TM can be written down as a long binary string
- \implies Canonical ordering of TMs:
 - $\{M_1, M_2, M_3, M_4, \dots, M_j, \dots\}$

The Diagonalization Language

- $L_d = \{ w_i \mid w_i \notin L(M_i) \}$
- The language of all strings whose corresponding machine does *not* accept itself (i.e., its own code)




- Table: $T[i,j] = 1$, if M_i accepts w_j
 $= 0$, otherwise.
- Make a new language called $L_d = \{ w_i \mid T[i,i] = 0 \}$

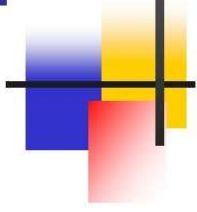


L_d is not RE (i.e., has no TM)

- Proof (by contradiction):
- Let M be the TM for L_d
- $\implies M$ has to be equal to some M_k s.t.
$$L(M_k) = L_d$$
- \implies Will w_k belong to $L(M_k)$ or not?
 1. If $w_k \in L(M_k) \implies T[k,k]=1 \implies w_k \notin L_d$
 2. If $w_k \notin L(M_k) \implies T[k,k]=0 \implies w_k \in L_d$
- A contradiction either way!!

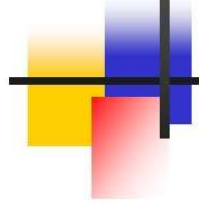


Why should there be
languages that do not have
TMs?

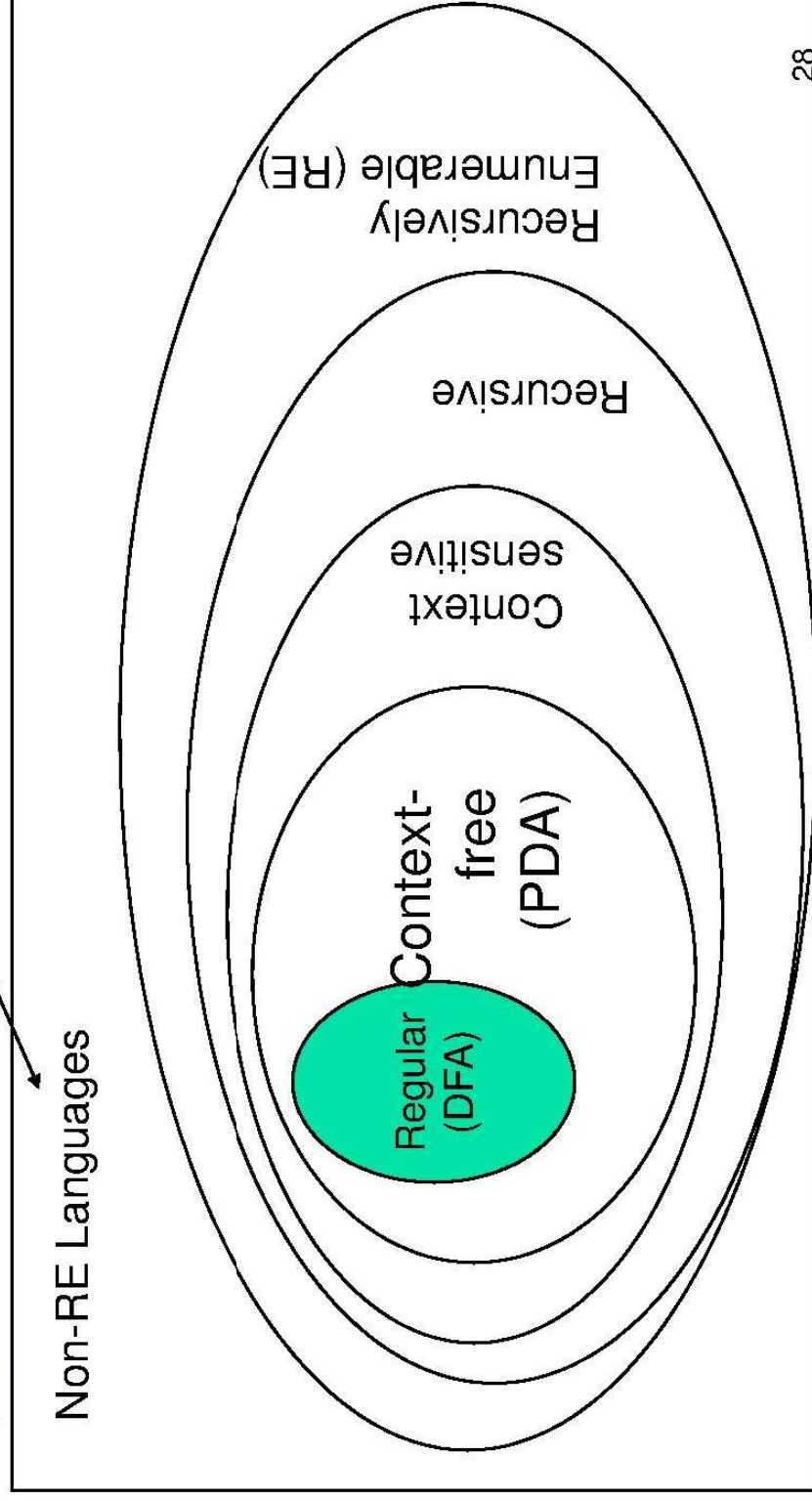


We thought TMs can solve
everything!!

Non-RE languages



How come there are languages here?
(e.g., diagonalization language)






One Explanation

There are more languages than TMs

- By pigeon hole principle:
 - \implies some languages cannot have TMs
- But how do we show this?
- Need a way to “count & compare” two infinite sets (languages and TMs)



How to count elements in a set?

Let A be a set:

- If A is finite \implies counting is trivial
- If A is infinite \implies how do we count?
- And, how do we compare two infinite sets by their size?

Cantor's definition of set "size" for infinite sets (1873 A.D.)

Let $N = \{1, 2, 3, \dots\}$ (all natural numbers)

Let $E = \{2, 4, 6, \dots\}$ (all even numbers)

Q) Which is bigger?

- A) Both sets are of the same size
 - "Countably infinite"
 - Proof: Show by one-to-one, onto set correspondence from $N \implies E$

n	$f(n)$
1	2
2	4
3	6
.	.
.	.

i.e., for every element in N , there is a unique element in E , and vice versa.

Really, really big sets!
(even bigger than countably infinite sets)

Uncountable sets

Example:


- Let \mathbb{R} be the set of all real numbers
- Claim: \mathbb{R} is uncountable

n	$f(n)$	
1	3. <u>1</u> 4 1 5 9 ...	Build x s.t. x cannot possibly occur in the table E.g. $x = 0.2644\dots$
2	5. 5 <u>5</u> 5 5 5 ...	
3	0. 1 2 <u>3</u> 4 5 ...	
4	0. 5 1 4 <u>3</u> 0 ...	
.	.	
.	.	
.	.	

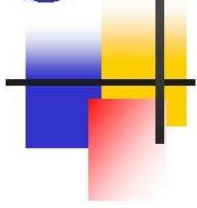



Therefore, some languages cannot have TMs...

- The set of all TMs is countably infinite
- The set of all Languages is uncountable
- \implies There should be some languages without TMs (by PHP)



The problem reduction technique, and reusing other constructions



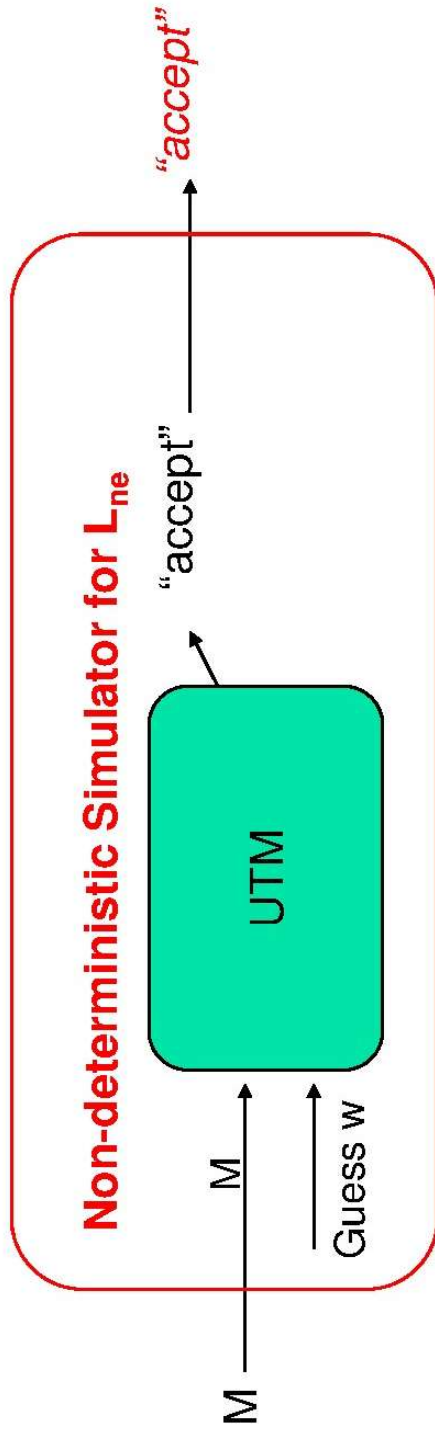


Languages that we know about

- *Language of a Universal TM (“UTM”)*
 - $L_U = \{ \langle M, w \rangle \mid M \text{ accepts } w \}$
 - Result: L_U is in RE but not recursive
- *Diagonalization language*
 - $L_d = \{ w_i \mid M_i \text{ does not accept } w_i \}$
 - Result: L_d is non-RE

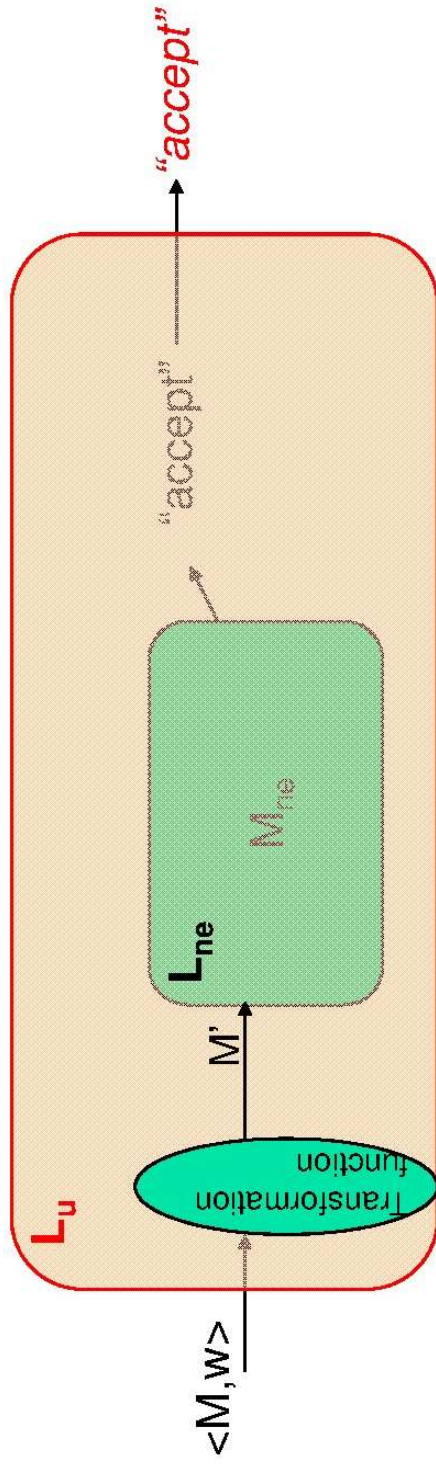
TMs that accept non-empty languages

- $L_{ne} = \{ M \mid L(M) \neq \emptyset \}$
- L_{ne} is RE
- Proof: (build a TM for L_{ne} using UTM)



TMs that accept non-empty languages

- L_{ne} is not recursive
- Proof: (“Reduce” L_u to L_{ne})
- Idea: M accepts w if and only if $L(M') \neq \emptyset$





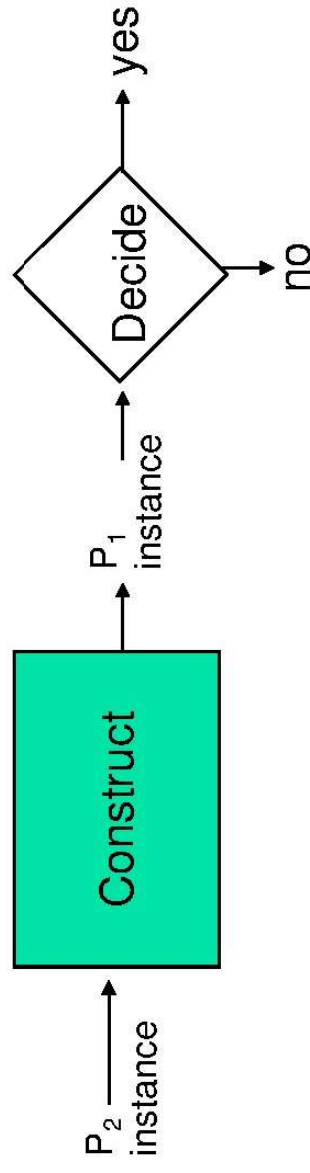
Reducability

- To prove: Problem P_1 is undecidable
- Given/known: Problem P_2 is undecidable
- Reduction idea:
 1. “Reduce” P_2 to P_1 :
 - Convert P_2 's input instance to P_1 's input instance s.t.
 - P_2 decides only if P_1 decides
 2. Therefore, P_2 is decidable
 3. A contradiction
 4. Therefore, P_1 has to be undecidable



The Reduction Technique

Reduce P_2 to P_1 :



Note:
not same as
 $P_1 \implies P_2$

Conclusion: If we could solve P_1 , then we can solve P_2 as well



Summary

- Problems vs. languages
- Decidability
 - Recursive
- Undecidability
 - Recursively Enumerable
 - Not RE
 - Examples of languages
- The diagonalization technique
- Reducability