Undecidability

Reading: Chapter 8 & 9

Decidability vs. Undecidability

- There are two types of TMs (based on halting):

  (*Recursive*)

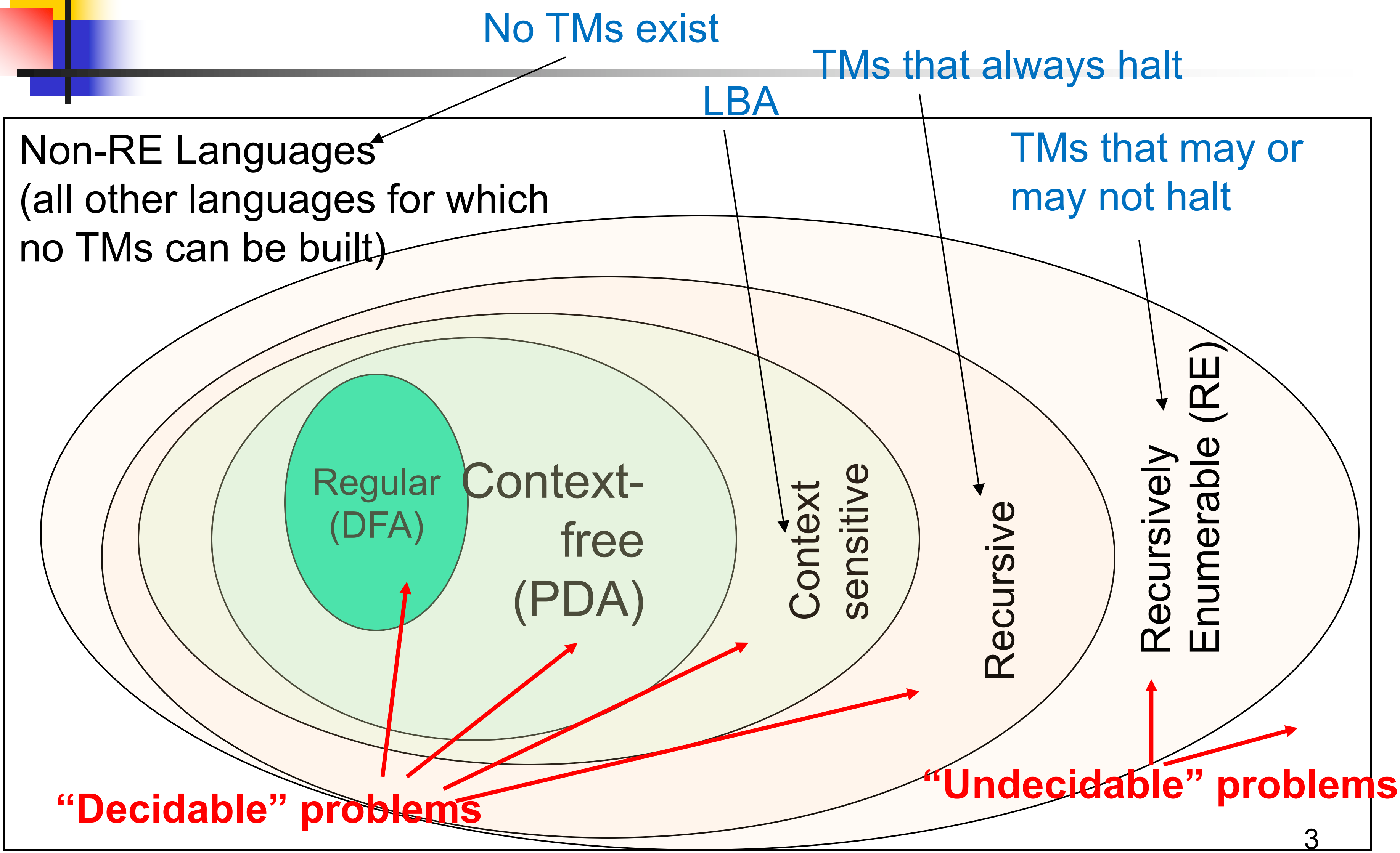
  TMs that *always* halt, no matter accepting or non-accepting ≡ **DECIDABLE** PROBLEMS

  (*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).

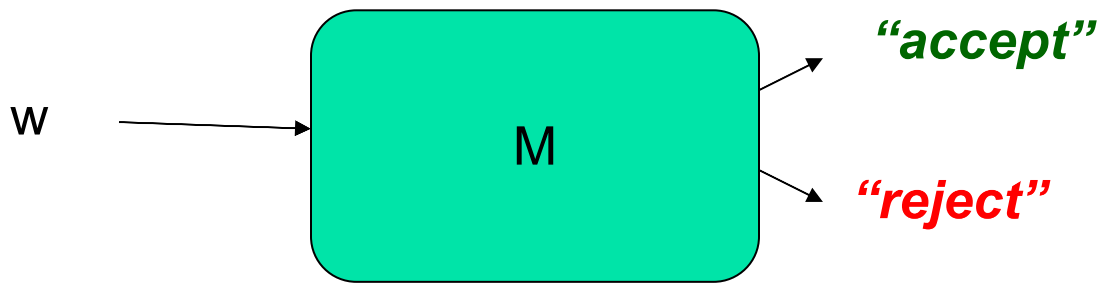
- **Undecidability:**
  - Undecidable problems are those that are not recursive

Recursive, RE, Undecidable languages



Recursive Languages & Recursively Enumerable (RE) languages

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

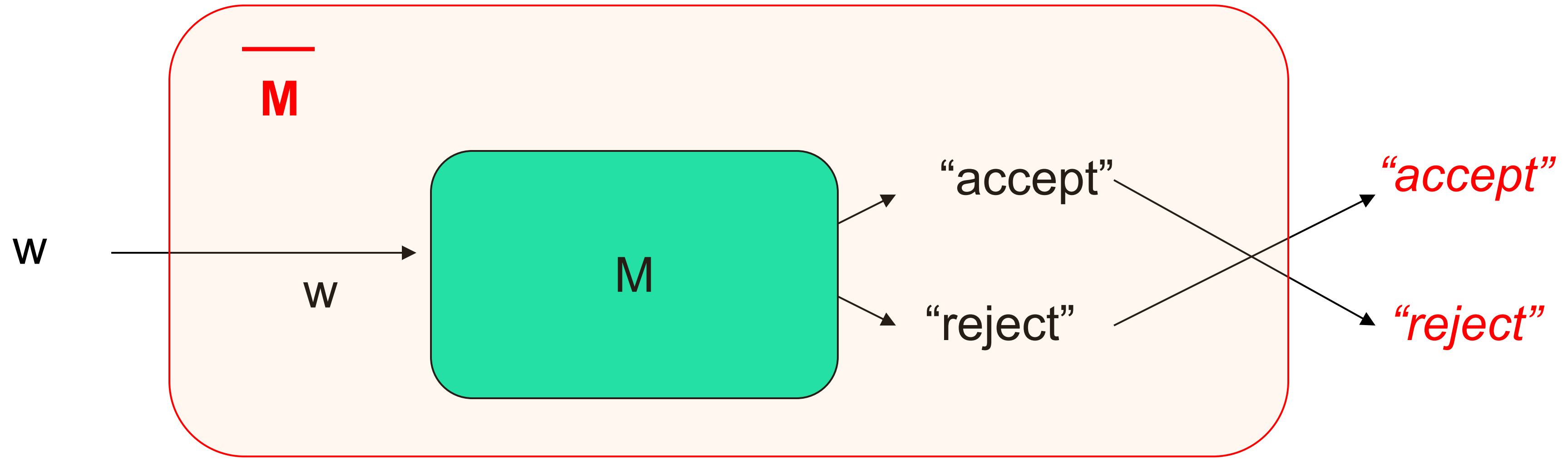


- Any TM for a Recursively Enumerable (RE) language is going to look like this:

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

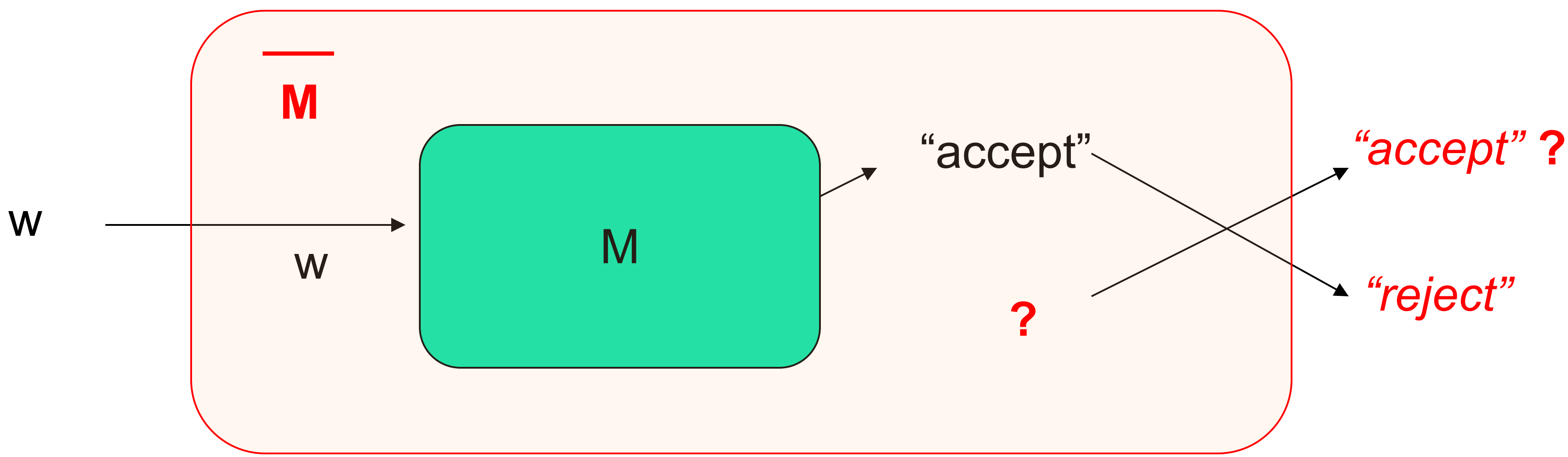
Recursive Languages are closed under complementation

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



Are Recursively Enumerable Languages closed under complementation? (NO)

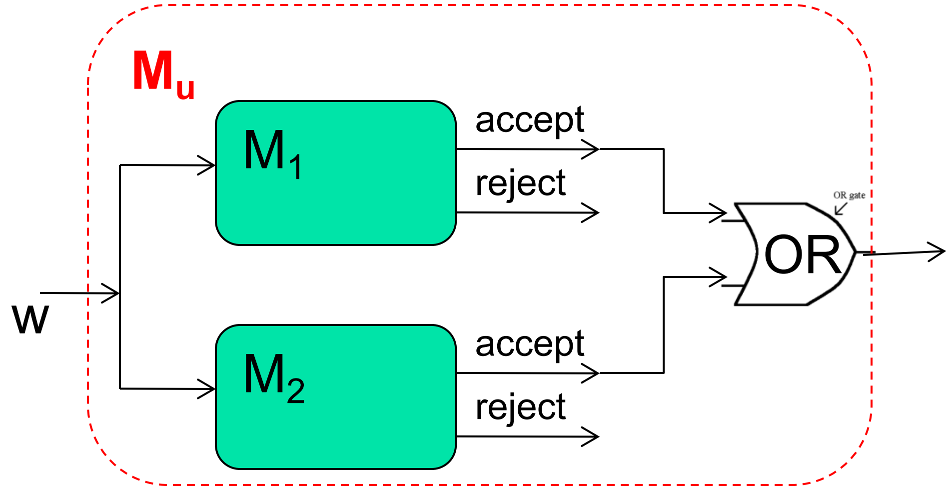
- If L is RE, $\overline{L}$ need not be RE



Recursive Langs are closed under Union

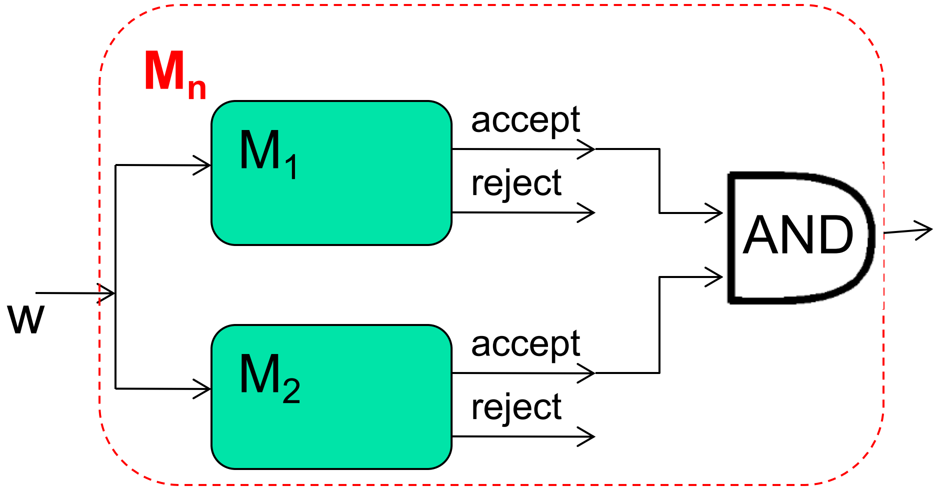
- Let M_u = TM for L_1 U L_2
- M_u construction:

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 Langs are closed under Intersection

- Let M_n = 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 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>”

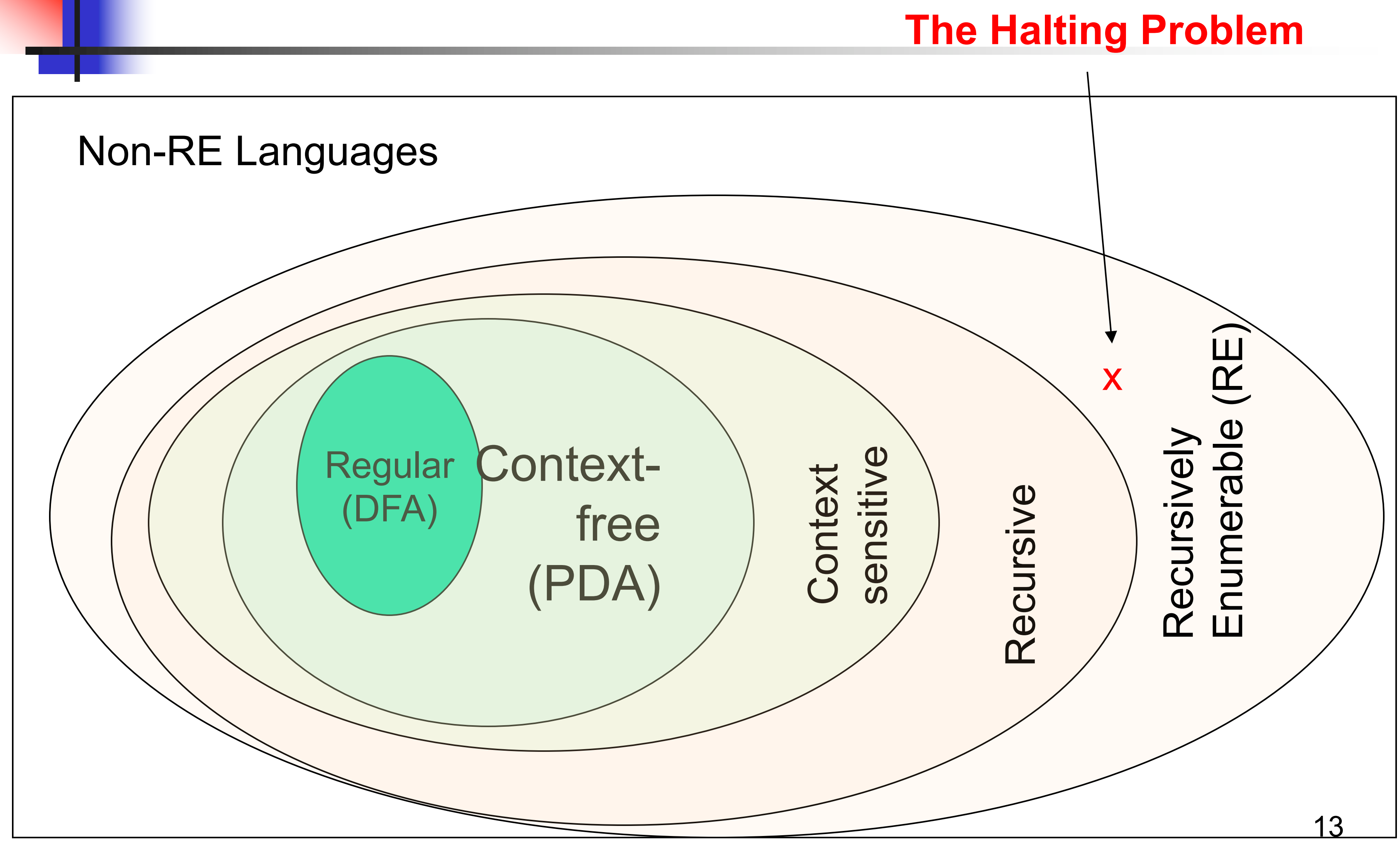
e.g., Problem (a+b) ≡ Language of strings of the form { “a#b, a+b” }

==> Every problem also corresponds to a language!!

Think of the language for a “problem” == a *verifier* for the problem

The Halting Problem

An example of a <u>recursive enumerable</u> problem that is also <u>undecidable</u>

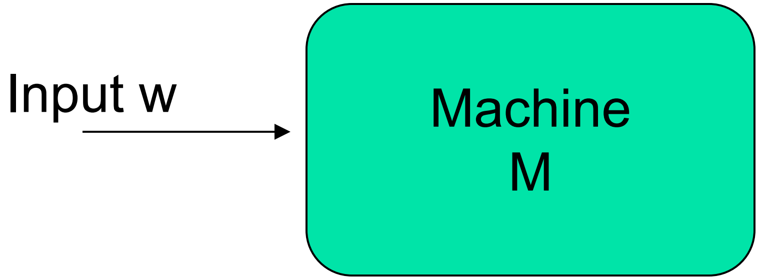


The Halting Problem
Non-RE Languages
x
Regular
(DFA)
Context-
free
(PDA)
Context
sensitive
Recursive
Recursively
Enumerable (RE)

What is the Halting Problem?

<u>Definition of the “halting problem”:</u>

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



A Turing Machine simulator

The Universal Turing Machine

- A universal Turing machine (UTM) is a Turing machine that simulates an arbitrary Turing machine on arbitrary input.
- 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

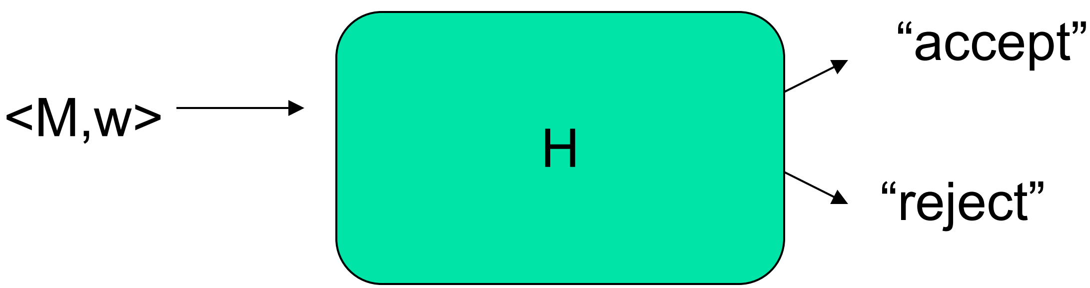
Implies: H is in RE

  - $$H(<M,w>) = \begin{cases} \textit{accept}, & \text{if } M \text{ accepts w} \\ \textit{reject}, & \text{if M does does not accept w} \end{cases}$$

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

A Claim

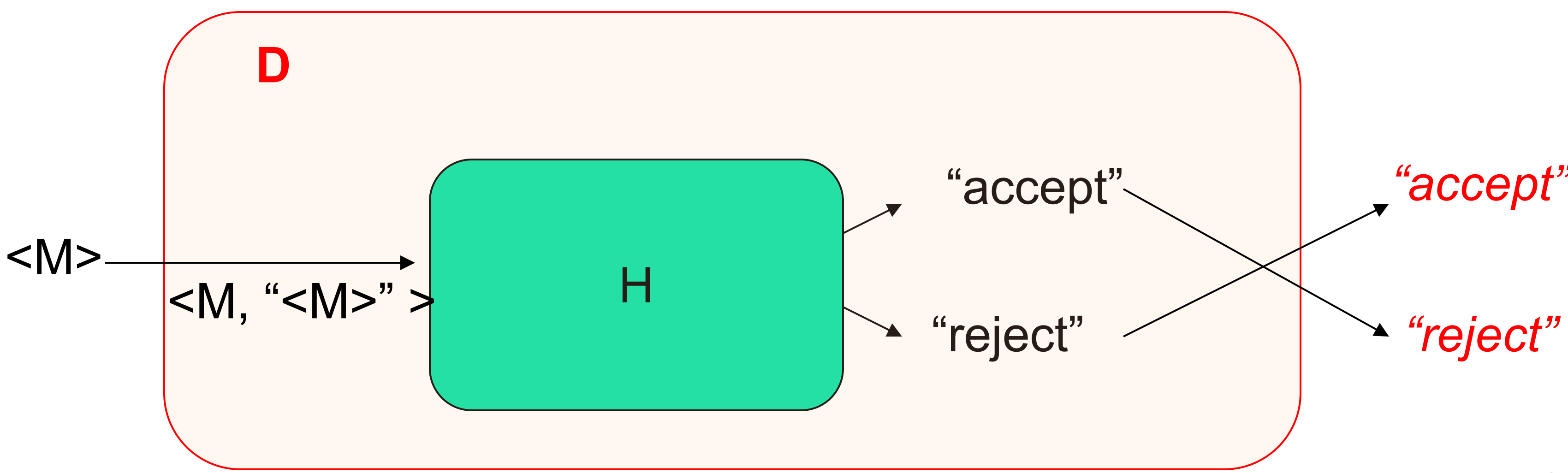
- <u>Claim:</u> No H that is always guaranteed to halt, can exist!
- <u>Proof:</u> (Alan Turing, 1936)
  - By contradiction, let us assume H exists



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 <M>:
    1. Run H on input <M, <M> >;   //(i.e., run M on M itself)
    2. Output the *opposite* of what H outputs;



HP Proof (step 2)

- The notion of inputing “<M>” to M itself
  - A program can be input to itself (e.g., a compiler is a program that takes any program as input)

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

Now, what happens if D is input to itself?

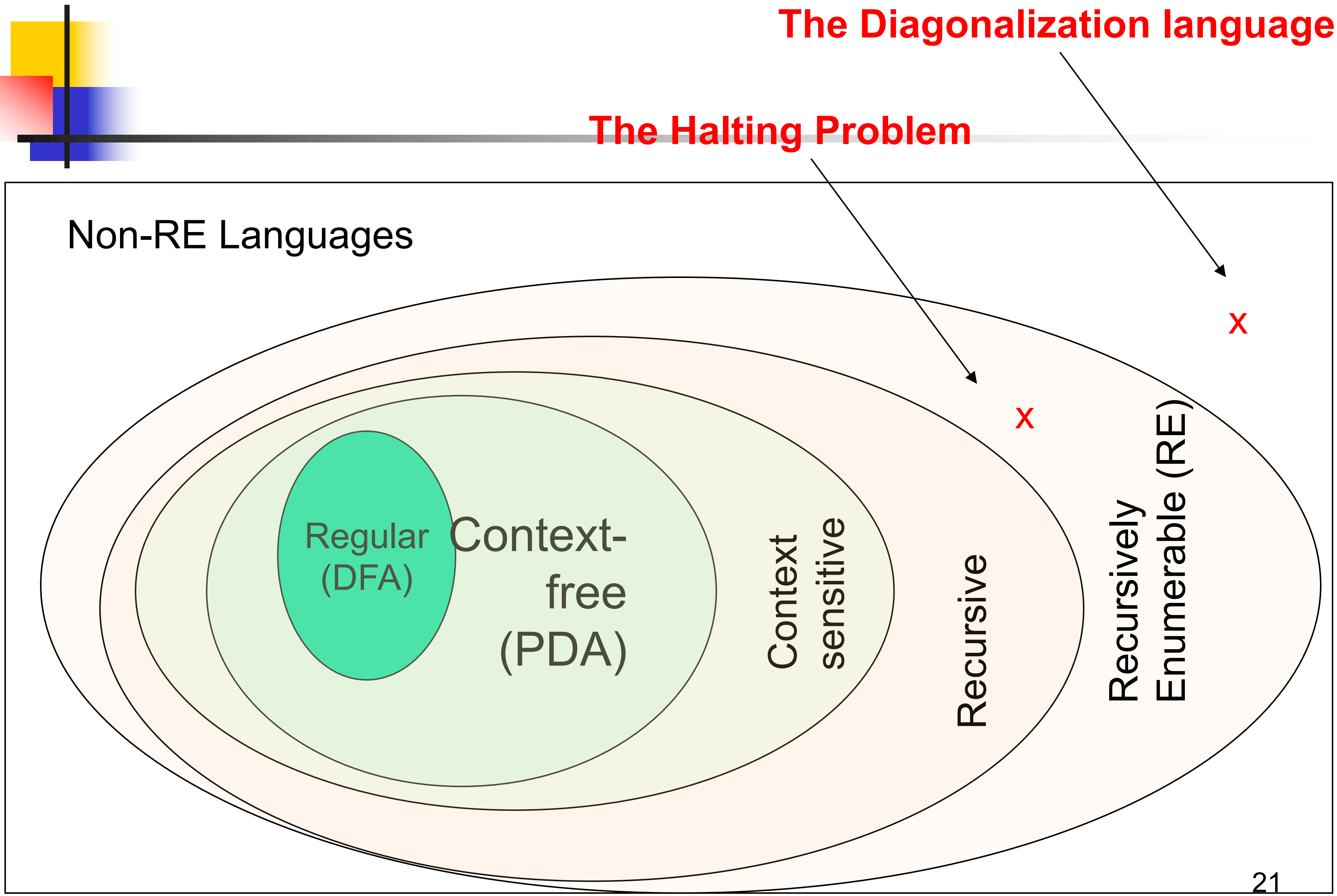
$$D\ (<D>) = \begin{cases} \textit{accept}, & \text{if D does not accept } <D> \\ \textit{reject}, & \text{if D accepts } <D> \end{cases}$$

A contradiction!!! ==> Neither D nor H can exist.

The Diagonalization Language

Example of a language that is
<u>not recursive enumerable</u>

(i.e, no TMs exist)


The Diagonalization language
The Halting Problem
Non-RE Languages
x
x
Regular
(DFA)
Context-
free
(PDA)
Context
sensitive
Recursive
Recursively
Enumerable (RE)

A Language about TMs & acceptance

- Let L be the language of all strings <M,w> 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=ε, 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.
- ***==> A <u>canonical ordering</u> of all binary strings:***
  - ***{ε, 0, 1, 00, 01, 10, 11, 000, 100, 101, 110, …..}***
  - ***{w_1, w_2, w_3, w_4, …. w_i, … }***

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 (==>binary strings)
  - $\delta(q_i,X_j) = (q_k,X_l,D_m)$ will be represented as:
    - ==> $0^i 1\ 0^j 1\ 0^k 1\ 0^l 1\ 0^m$
- <u>Result: </u>Each TM can be written down as a long binary string
- ==> Canonical ordering of TMs:
  - $\{M_1, M_2, M_3, M_4, \ldots .\ M_i, \ldots \}$

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)

(input word w)

j →

(TMs)

i ↓

	1	2	3	4	…
1	0	1	0	1	…
2	1	1	0	0	…
3	0	1	0	1	…
4	1	0	0	1	…
⋮					⋱

diagonal

- Table: T[i,j] = 1, if M_i accepts w_j
  = 0, otherwise.

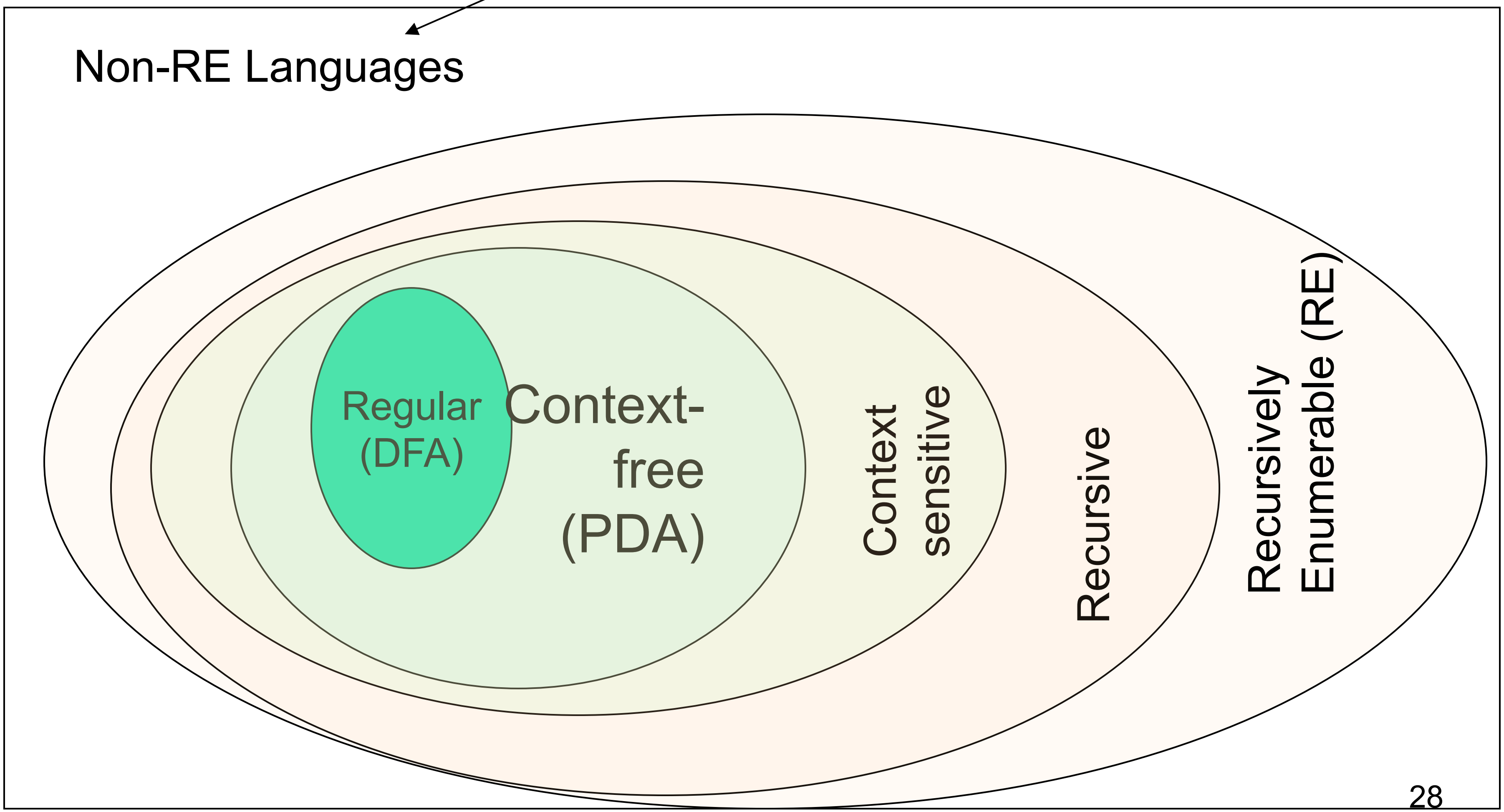
- Make a new language called
  $L_d = \{w_i \mid T[i,i] = 0\}$

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:
- ==> 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 ==> counting is trivial
- If A is infinite ==> 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,…}   (all natural numbers)

Let E = {2,4,6,…}   (all even numbers)

Q) Which is bigger?

- A) Both sets are of the same size
  - "***Countably infinite***"
  - <u>Proof:</u> Show by one-to-one, onto set correspondence from N ==> 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 R be the set of all real numbers
- Claim: R is uncountable

n	*f(n)*
1	3 . 1 4 1 5 9 …
2	5 . 5 5 5 5 5 …
3	0 . 1 2 3 4 5 …
4	0 . 5 1 4 3 0 …
.	
.	
.	

Build x s.t. x cannot possibly occur in the table

E.g. x = 0 . 2 6 4 4 …

Therefore, some languages cannot have TMs…

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

Summary

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