CS311: Computational Theory

Lecture 11: Decidability– Ch 4

Dr. Manal Helal, Spring 2014.

http://moodle.manalhelal.com

Lecture Learning Objectives

1. *Understand Computability Theory*

The HELLO WORLD assignment

- *Suppose your teacher tells you:*
    - *Write a JAVA program to output the word “HELLO WORLD” on the screen and halt.*
- *Space and time are not an issue.*
- *The program is for an ideal computer.*
- *PASS for any working HELLO program, no partial credit.*

Teacher’s Grading Program

- *The grading program G must be able to take any Java program P and grade it.*
- $$G(P)=\begin{cases}\text{Pass, if } P \text{ prints “HELLO WORLD”}\\ \text{Fail, otherwise.}\end{cases}$$

How exactly might such a script work?

What kind of program could a student who hated his/her teacher hand in?

Nasty Program

```
n:=2;
While (the number 2n can be written as the sum of two primes)
    n++;
Print “HELLO WORLD”;
```

Despite the simplicity of the HELLO WORLD assignment, there is no program

- *The nasty program is a PASS if and only if the Goldbach conjecture is false.*

Theory of Computation

- *The theory of computation studies*
    - *whether (computability theory or recursion theory),*
    - *and how efficiently (complexity theory)*
- *certain problems can be solved on a computer, or rather on a model of a computer.*
- *Several equivalent models of computational devices can be used:*
    - *Register machines.*
    - *Lambda calculus.*
    - *Simple while programming language (pseudo-code).*
    - *Turing machines.*
    - *...*
- *We focus on Turing machines.*

Computability

- *Do algorithmic solutions to problems always exist?*
- *What are the limitations of computational devices?*
- *Is there any insight to which problems are algorithmically solvable and which are not?*
- *Are the unsolvable problems somehow related?*

Complexity (CS312)

- *How do we compare the efficiency of different algorithms?*
- *How do we measure time/memory requirements?*
- *What problems are efficiently solvable?*
- *Are there solvable problems which do not have efficient algorithms?*

Decidability vs. Undecidability

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

  (Recursive)

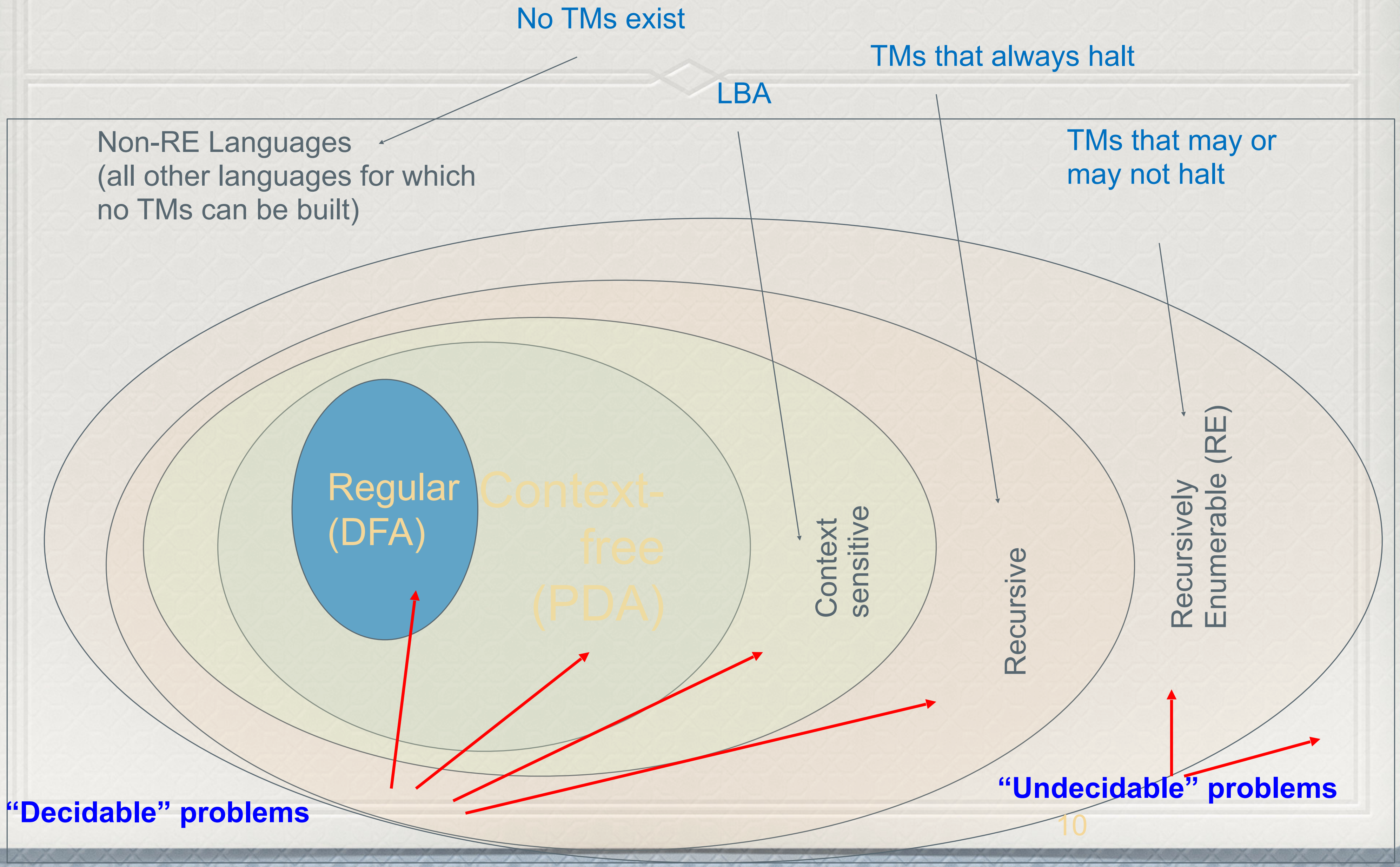
  ***TMs that always halt**, no matter accepting or non-accepting ≡*
  ***DECIDABLE** PROBLEMS*

  (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



No TMs exist

TMs that always halt

LBA

Non-RE Languages
(all other languages for which
no TMs can be built)

TMs that may or
may not halt

Regular
(DFA)

Context-
free
(PDA)

Context
sensitive

Recursive

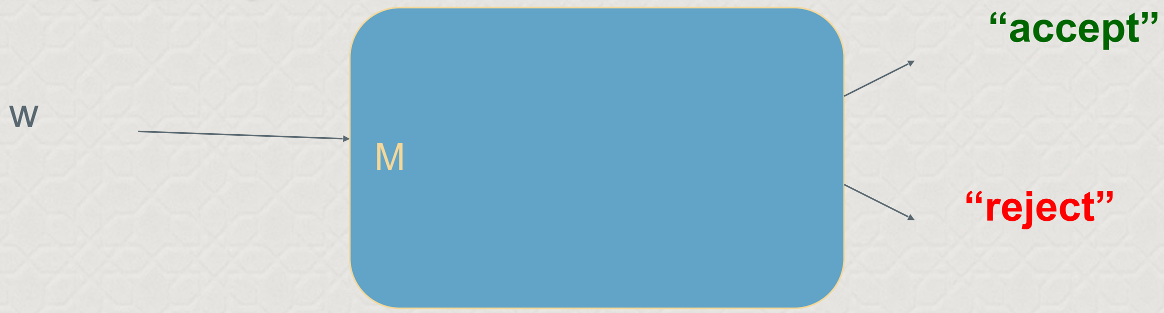
Recursively
Enumerable (RE)

“Undecidable” problems

“Decidable” problems

Recursive Languages & Recursively Enumerable (RE) languages

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



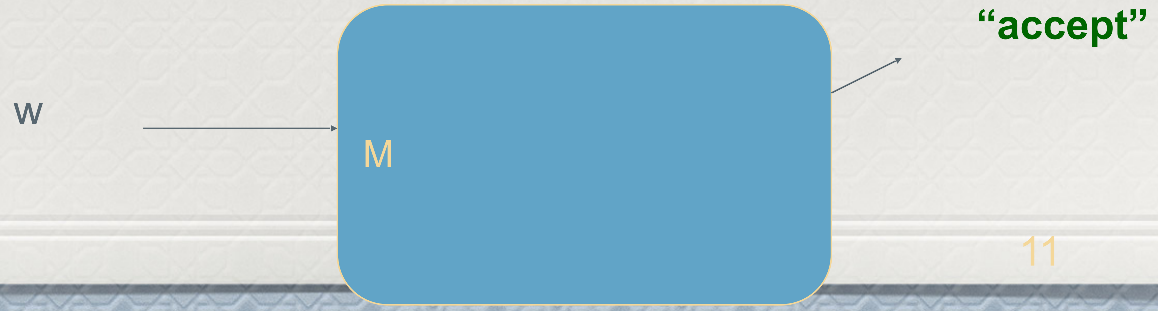
w

M

"accept"

"reject"

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



w

M

"accept"

Recognizable and Decidable Languages

- ***(Language of M):*** *The language recognized by a TM M, or simply the language of M is* $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$.
- ***(Recognizable Language):*** *A language* $L \subseteq \Sigma^*$ *is recognizable if there exists a TM M such that M recognizes L, i.e.,* $L = L(M)$.
- ***(Decidable Language):*** *A language* $L \subseteq \Sigma^*$ *is decidable if there exists a TM M such that M is a decider and M recognizes L, i.e.,* $L = L(M)$.

Example

- *Consider the language L = {$a^n b^n c^n | n \geq 0$}.*
- *Facts:*
    - *L is not regular,*
    - *L is not context-free, but*
    - *L is recognizable and even decidable language.*

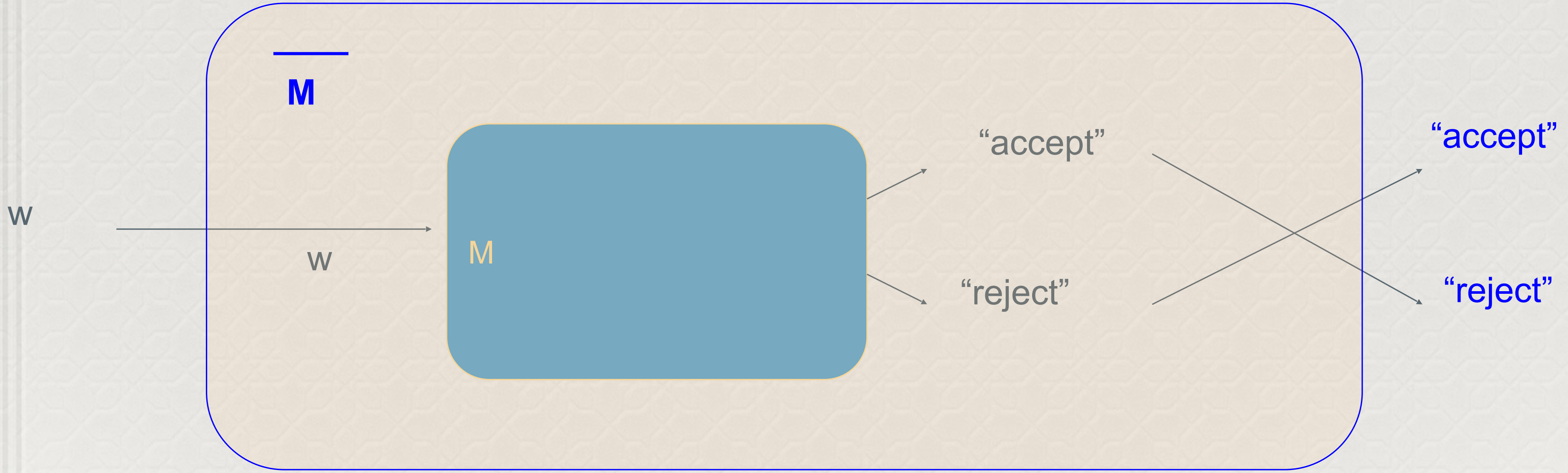
Exam Questions

- *Definition of a Turing machine, configuration, computation, acceptance of a string by a TM.*
- *Definition of a decider.*
- *Definition of recognizable and decidable languages.*

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

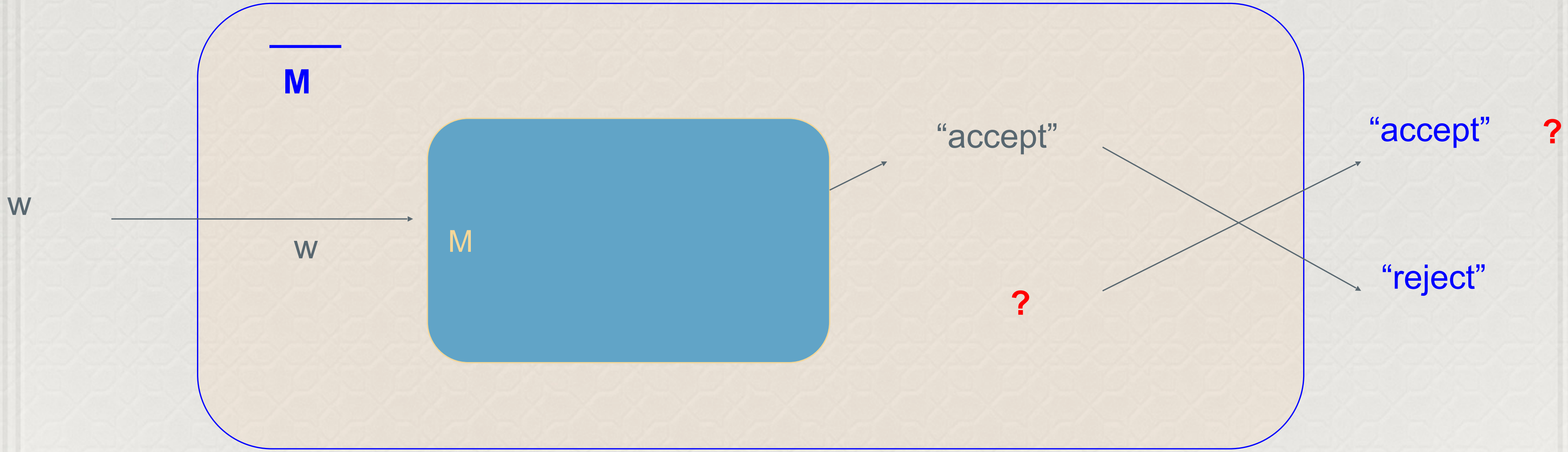
Recursive Languages are closed under complementation

- *If L is Recursive, L is also Recursive*



Are Recursively Enumerable Languages closed under <u>complementation</u>? (NO)

- *If L is RE, L need not be RE*



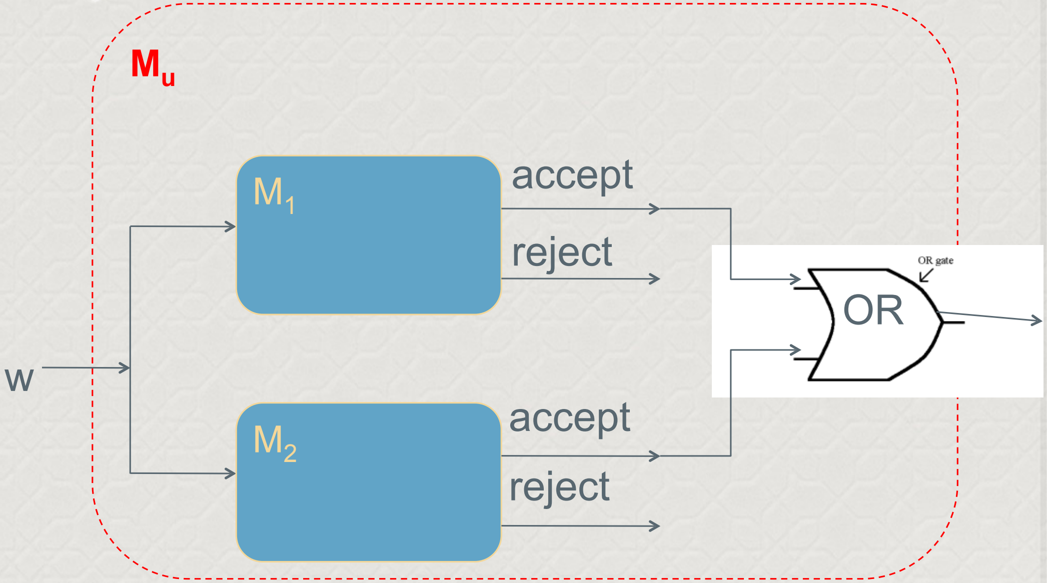
Recursive Langs are closed under Union

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*

- *Otherwise, M_u rejects.*



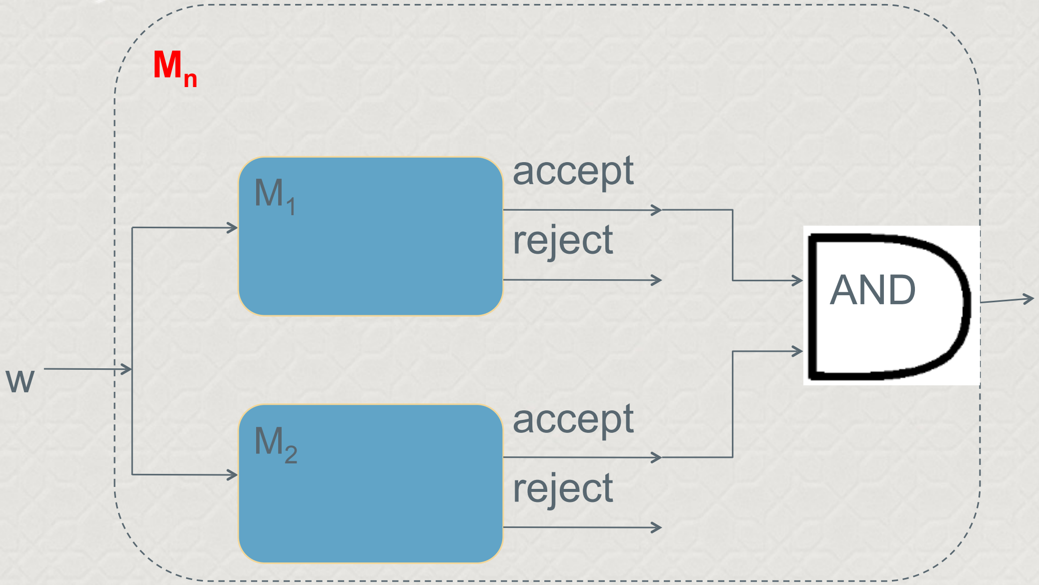
Recursive Langs are closed under Intersection

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 either M_1 AND M_2 accepts, then M_n accepts*

– *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

Computable Functions

- *Fix any precise programming language, i.e., Java.*
- *A program is any finite string of symbols from Σ that a Java interpreter will run (won't give a syntax error)*
- *Recall Σ* is the set of all strings of symbols.*

- *A function f : Σ* -> Σ* is computable if there is a program P that computes f, when P is executed on a computer with ideal memory.*

- *That is, for all strings x in Σ*, P(x) = f(x).*

Computable Functions - Cont’d

- *The set of all* **programs** *is a* **countable set***!*
- *The set of all* **computable functions** *is also a* **countable set!**
- *Are there countably many functions from* Σ^* *to* Σ^* *?*

Power Sets

- *Let S be a set.*
- *The power set of S is the set of all subsets of S.*

- *We write the power set as Power(S).*

- *Proposition: If S is finite, then Power(S) has cardinality* $2^{|S|}$

Theorem: For every S, there is no bijection between S and Power(S)

- *Suppose $f: S \rightarrow \prod(S)$ is 1-1 and ONTO.*
- *Let WEIRD = $\{ x \mid x \in S, x \notin f(x) \}$ There's some y in S such that f(y)=WEIRD*
- *Is y in WEIRD? YES or NO?*
- *if y in WEIRD, then $y \in S$, and $y \notin f(y)$ = WEIRD*
- *So y is not in WEIRD... but then*
  $y \in S$ and $y \notin$ WEIRD = f(y)... So y is in WEIRD...

Contradiction

Theorem: There are uncountably many functions!

- *There is a bijection between - The set of all subsets of Σ^* (the powerset of Σ^*)*
- *The set of all functions f: Σ^* -> {0,1} Take a subset S of Σ^*, we map it to the*
- *function f where:*

  f(x) = 1 (x in S), f(x) = 0 (x not in S)
- *So the set of all f: Σ^* -> {0,1} has the same size as the powerset of Σ^**
- *But Σ^* is countable, so the powerset of Σ^* is uncountable! (No bijection between Σ^* and Power(Σ^*)!)*

So there are functions from Σ^* to $\{0,1\}$ that are not computable.

Can we describe an incomputable one? Can we describe an interesting, incomputable function?

Notation And Conventions

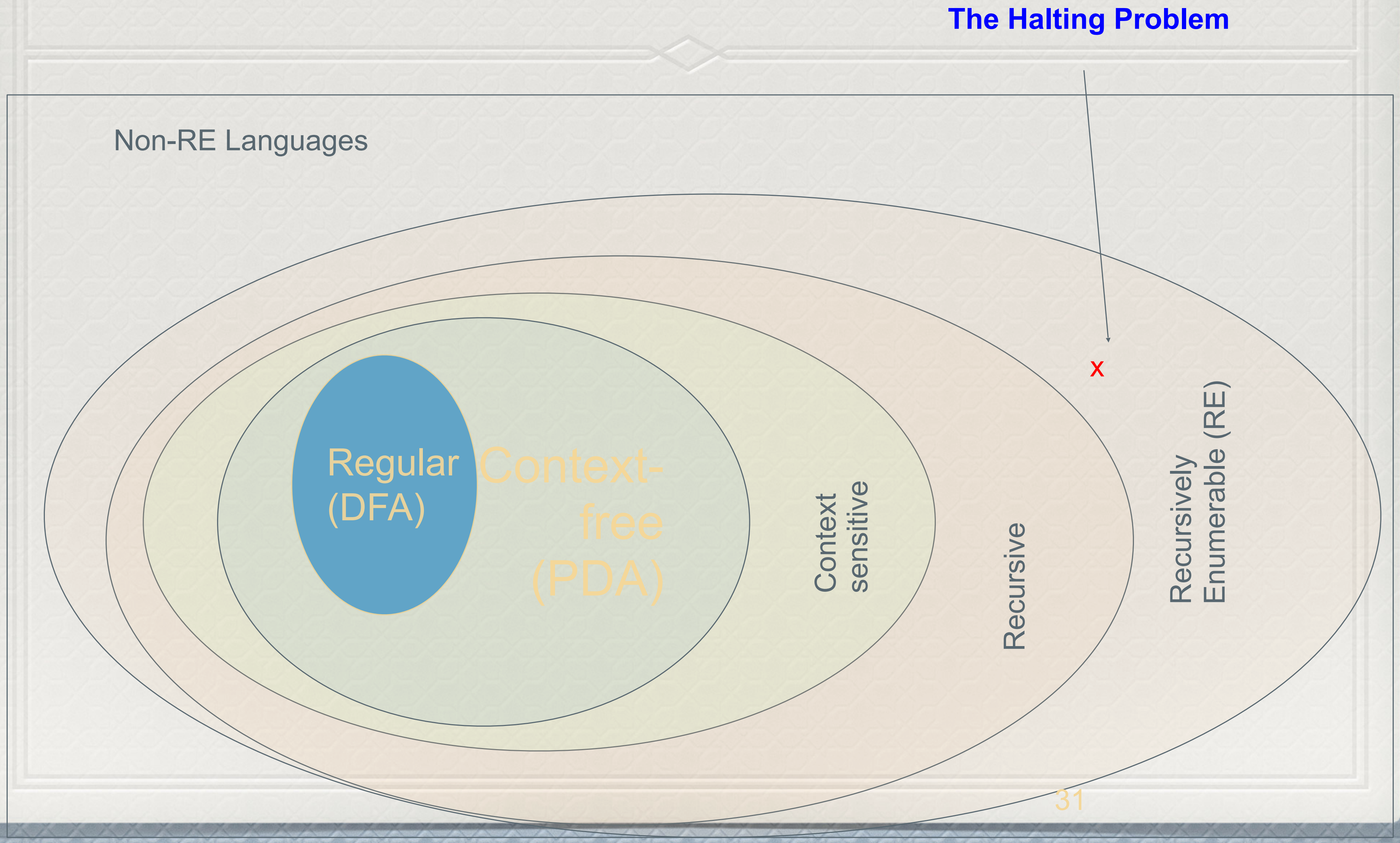
- *Fix any programming language*
- *When we refer to “program P” we mean the text of the source code for P*
- *P(x) is the final output of program P on input x, assuming that P eventually halts*

P(P)

- *It follows from our conventions that P(P) is the output obtained when we run P on the text of its own source code.*

Example 1: The Halting Problem

An example of a recursive enumerable problem that is also undecidable



The Halting Problem

Non-RE Languages

x

Regular (DFA)

Context-free (PDA)

Context sensitive

Recursive

Recursively Enumerable (RE)

The Famous Halting Set: K

- *K is the set of all programs P such that P(P) halts.*
- *K = { Program P | P(P) halts}*

The Halting Problem

- *Is there a program HALT such that:*

- *HALT(P) = yes, if P(P) halts*
- *HALT(P) = no, if P(P) does not halt*

The Halting Problem K = {P | P(P) halts }

- *Is there a program HALT such that:*

- *HALT(P) = yes, if P∈K*
- *HALT(P) = no, if P∉K*

- *HALTS decides whether or not any given program is in K.*

THEOREM: There is no program that can solve the halting problem! (Alan Turing 1937)

- *Suppose a program HALT, solving the halting problem, existed:*

- *HALT(P) = yes, if P(P) halts*
- *HALT(P) = no, if P(P) does not halt*

- *We will call HALT as a subroutine in a new program called WEIRD.*

The Program WEIRD(P):

If HALT(P) then go into an infinite loop.
Else stop.
<Put text of subroutine HALT here>

Contradiction

- *Does WEIRD(WEIRD) halt or not?*
- *YES implies HALT(WEIRD) = yes*

but then, WEIRD(WEIRD) will infinite loop

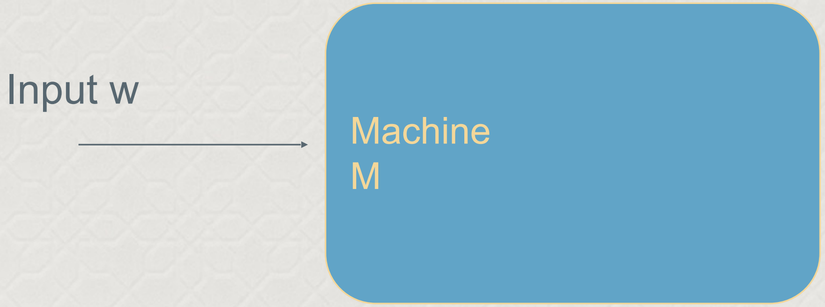
- *NO implies HALT(WEIRD) = no but then,*

WEIRD(WEIRD) halts

What is the Halting Problem?

Definition of the “halting problem”:

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



Input w

Machine M

A Turing Machine simulator

The Universal Turing Machine

- *__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*

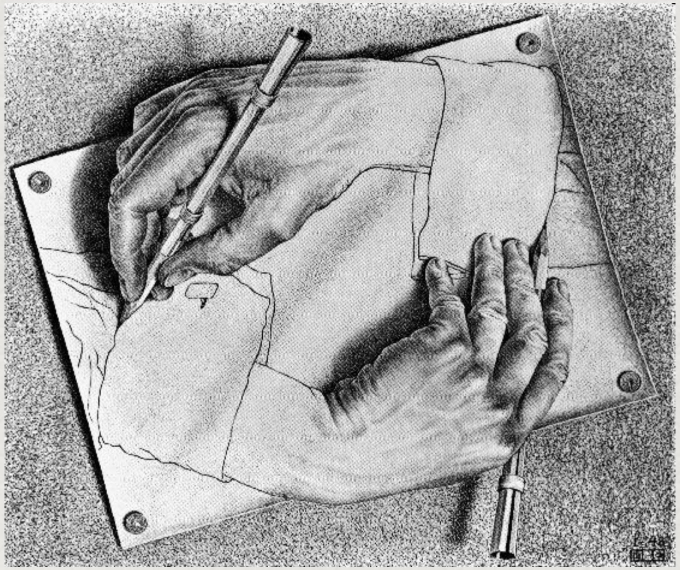
Implies: H is in RE

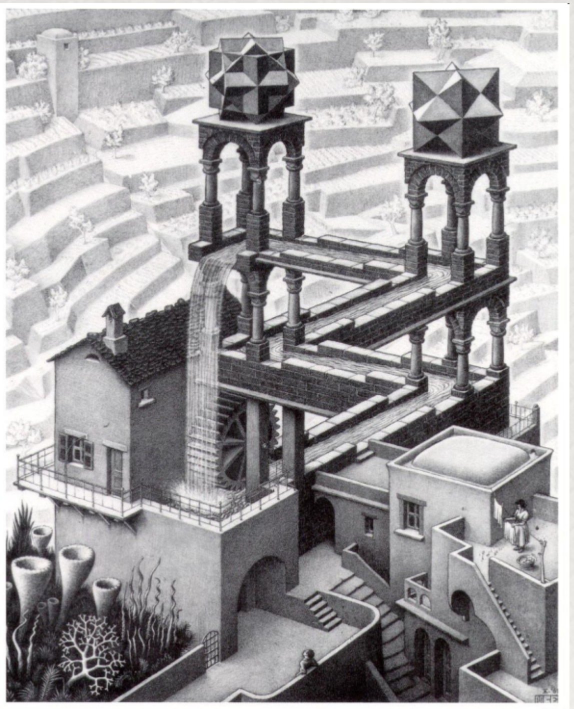
- $$H(\langle M,w\rangle) = \begin{cases} \text{accept}, & \text{if M accepts w} \\ \text{reject}, & \text{if M does does not accept w} \end{cases}$$

__Question:__ If M does not halt on w, what will happen to H?

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:

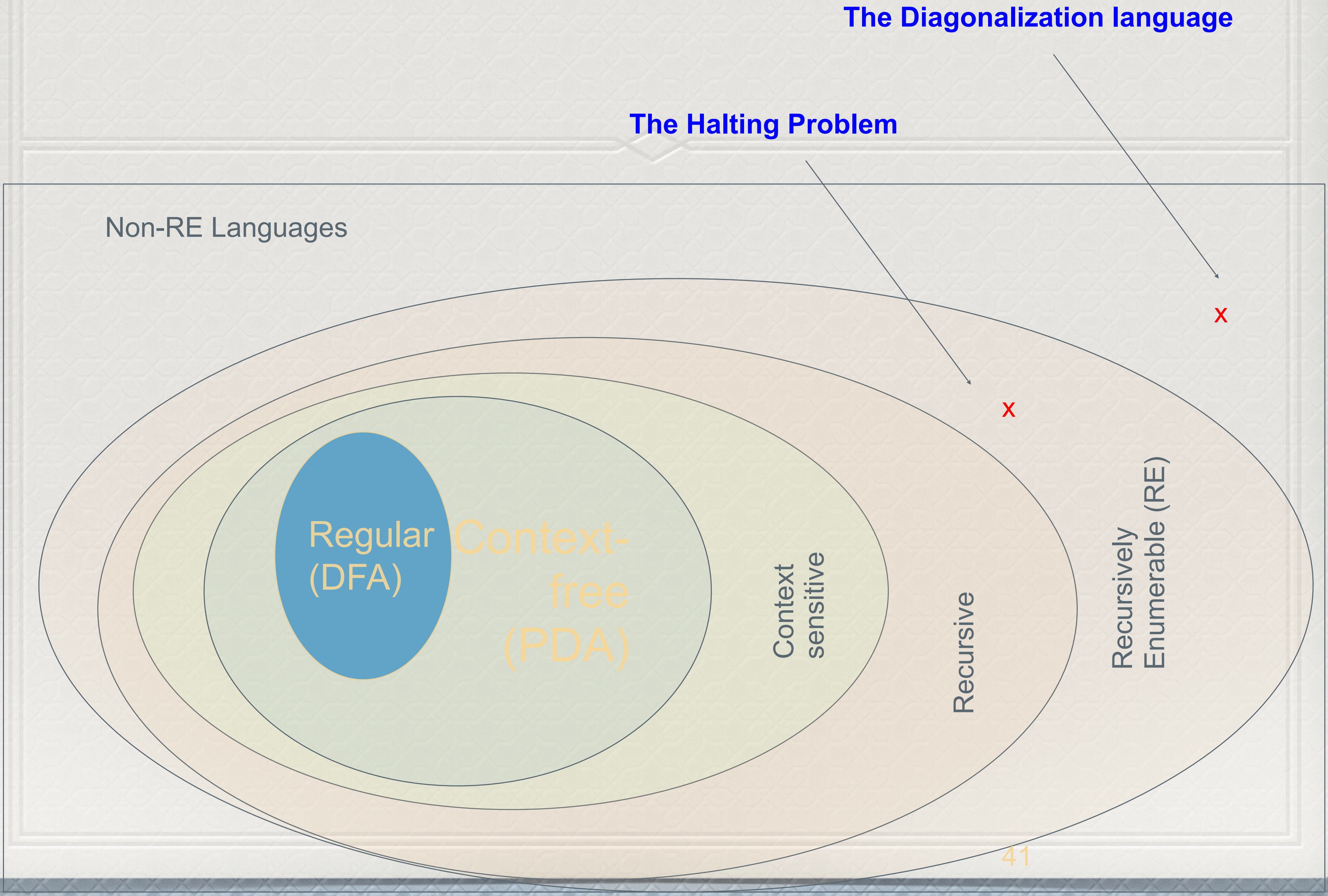
"Godel, Escher, Bach: An Eternal Golden Braid"
by Douglas Hofstadter (Pulitzer winner, 1980)

Turing's argument is just like the DIAGONALIZATION argument from the theory of infinities.

Example 2: The Diagonalization Language

Example of a language that is not recursive enumerable

(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 ≡ 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≡ 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, … }***

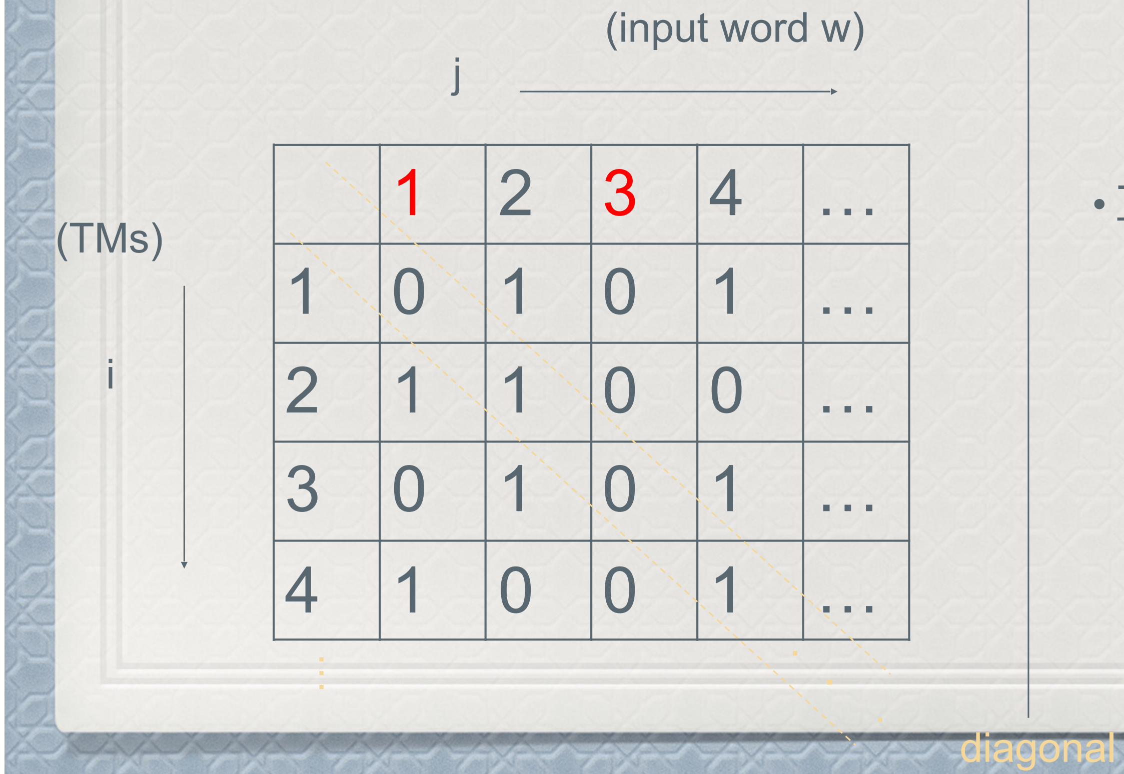
Any TM M can also be binary-coded

- $M = \{ Q, \{0,1\}, \Gamma, \delta, q_0, B, F \}$
  - 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:
    - ==> $0^i1\ 0^j1\ 0^k1\ 0^l1\ 0^m$

- *Result:* 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
diagonal

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

- 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*
- *==> M has to be equal to some M_k s.t.*

  $L(M_k) = L_d$
- *==> Will w_k belong to $L(M_k)$ or not?*

1. *If $w_k \in L(M_k)$ ==> T[k,k]=1 ==> $w_k \notin L_d$*

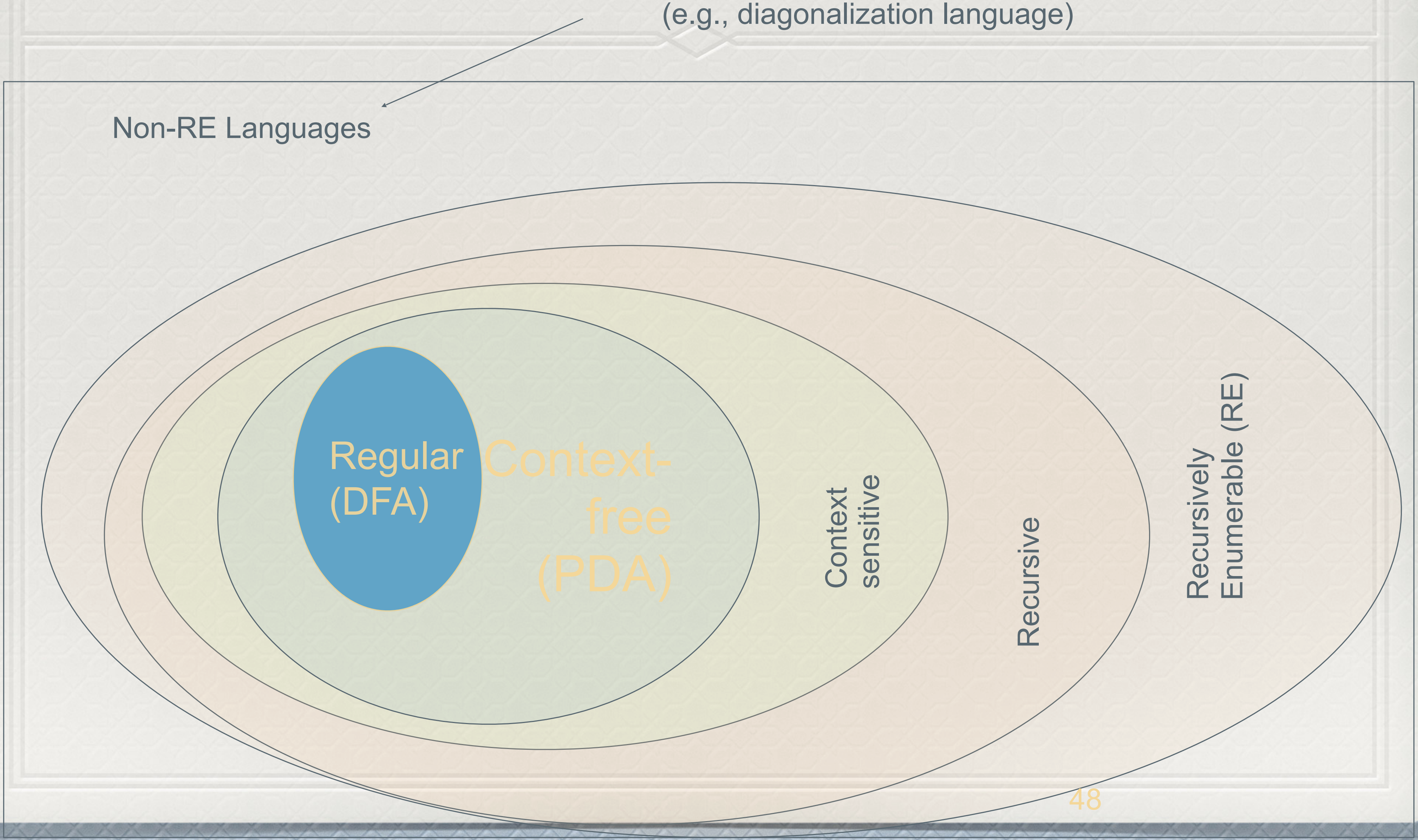
– *If $w_k \notin L(M_k)$ ==> T[k,k]=0 ==> $w_k \in L_d$*

- *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)

Non-RE Languages

Regular
(DFA)

Context-
free
(PDA)

Context
sensitive

Recursive

Recursively
Enumerable (RE)

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?

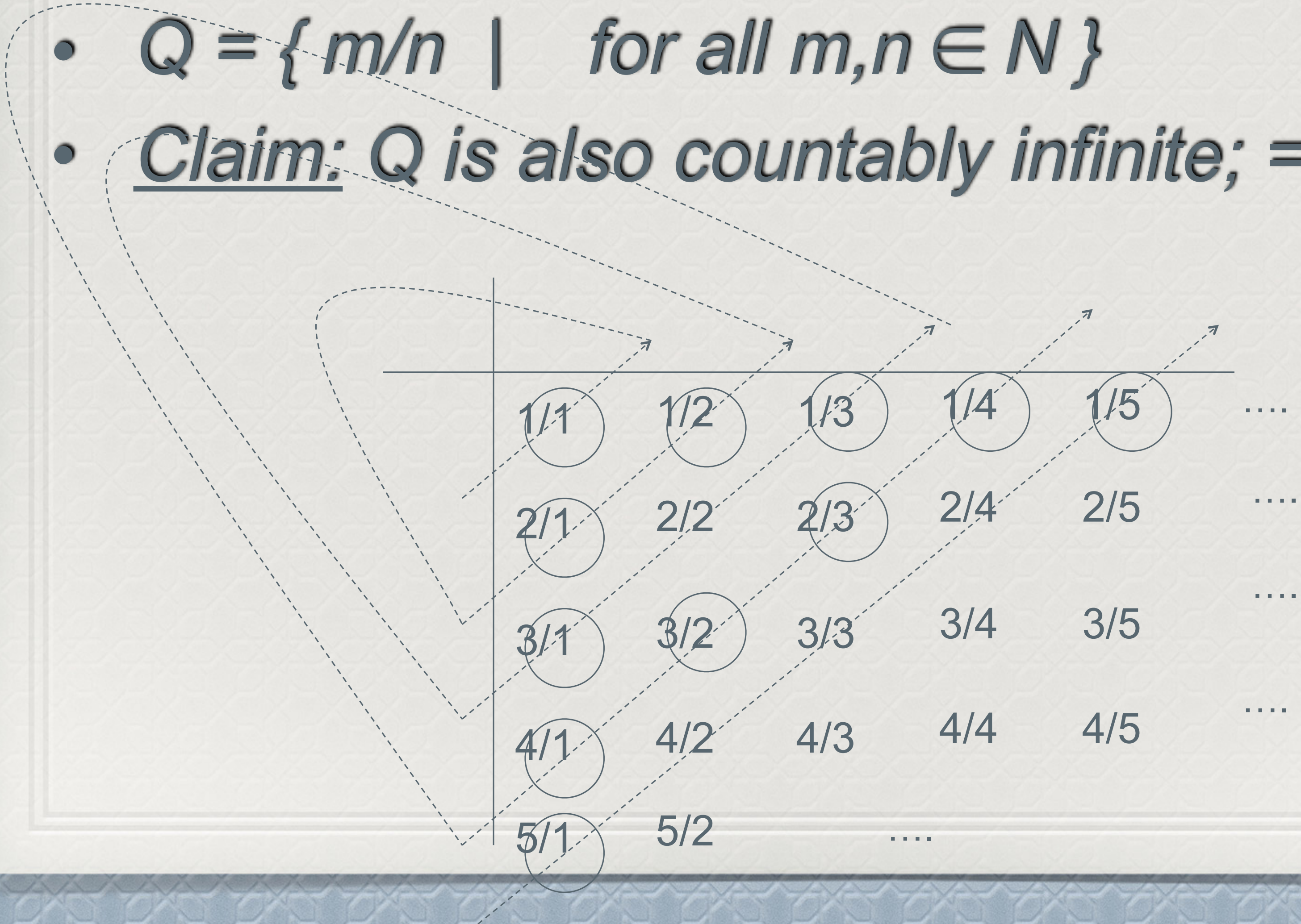
- *A) Both sets are of the same size*
  - ○ *"**Countably infinite**"*
  - ○ *Proof: Show by one-to-one, onto set correspondence from N ==> E*

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.

Example #3

- *Let Q be the set of all rational numbers*
- *$Q = \{ m/n \mid \text{for all } m,n \in N \}$*
- ***Claim:*** *Q is also countably infinite; => |Q|=|N|*



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	....			

Uncountable sets

Example:

- Let R be the set of all real numbers
- Claim: R is uncountable

n	f(n)
1	$3\ .\ \underline{1}\ 4\ 1\ 5\ 9\ \ldots$
2	$5\ .\ 5\ \underline{5}\ 5\ 5\ 5\ \ldots$
3	$0\ .\ 1\ 2\ \underline{3}\ 4\ 5\ \ldots$
4	$0\ .\ 5\ 1\ 4\ \underline{3}\ 0\ \ldots$
.	
.	
.	

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

E.g. $x = 0\ .\ 2\ 6\ 4\ 4\ \ldots$

Other Non-Computable Functions

- *Concrete examples of finitary functions are Busy beaver, Kolmogorov complexity, or any function that outputs the digits of a noncomputable number, such as Chaitin's constant.*

- *Similarly, most subsets of the natural numbers are not computable. The Halting problem was the first such set to be constructed.*
- *The Entscheidungsproblem, proposed by David Hilbert, asked whether there is an effective procedure to determine which mathematical statements (coded as natural numbers) are true.*
    - *Turing and Church independently showed in the 1930s that this set of natural numbers is not computable. According to the Church–Turing thesis, there is no effective procedure (with an algorithm) which can perform these computations.*

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)*

The problem reduction technique, and reusing other constructions

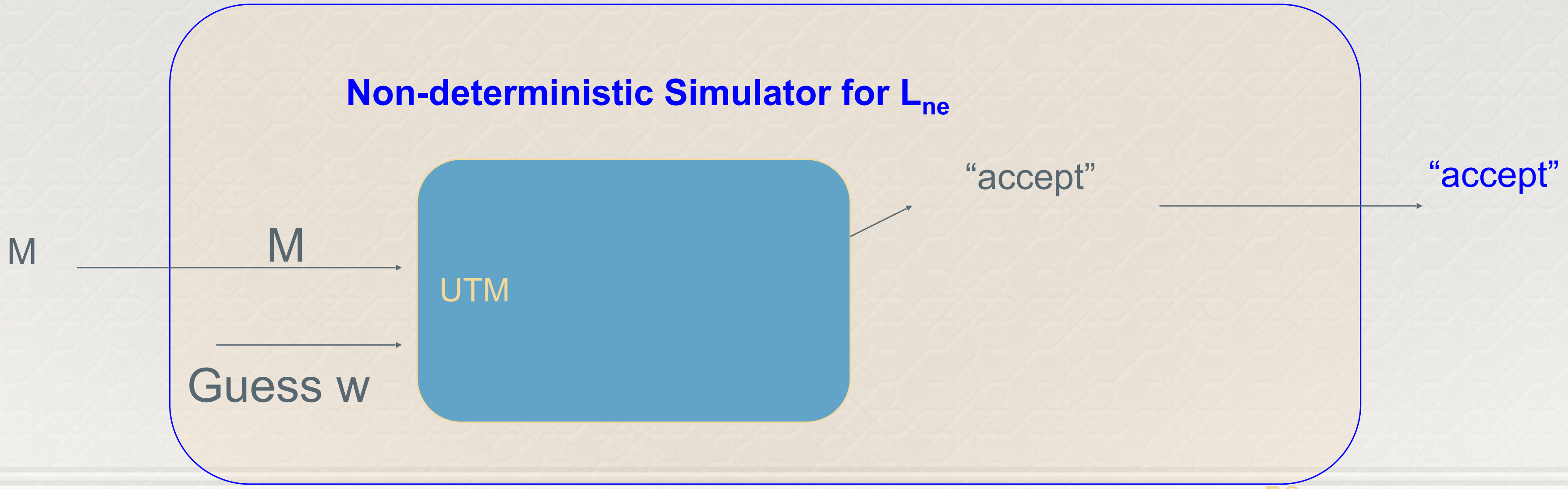
Languages that we know about

- *Language of a Universal TM ("UTM")*
    - *L_u = { <M,w> | M accepts w }*
    - *<u>Result:</u> L_u is in RE but not recursive*

- *Diagonalization language*
    - *L_d = { w_i | M_i does not accept w_i }*
    - *<u>Result:</u> L_d is non-RE*

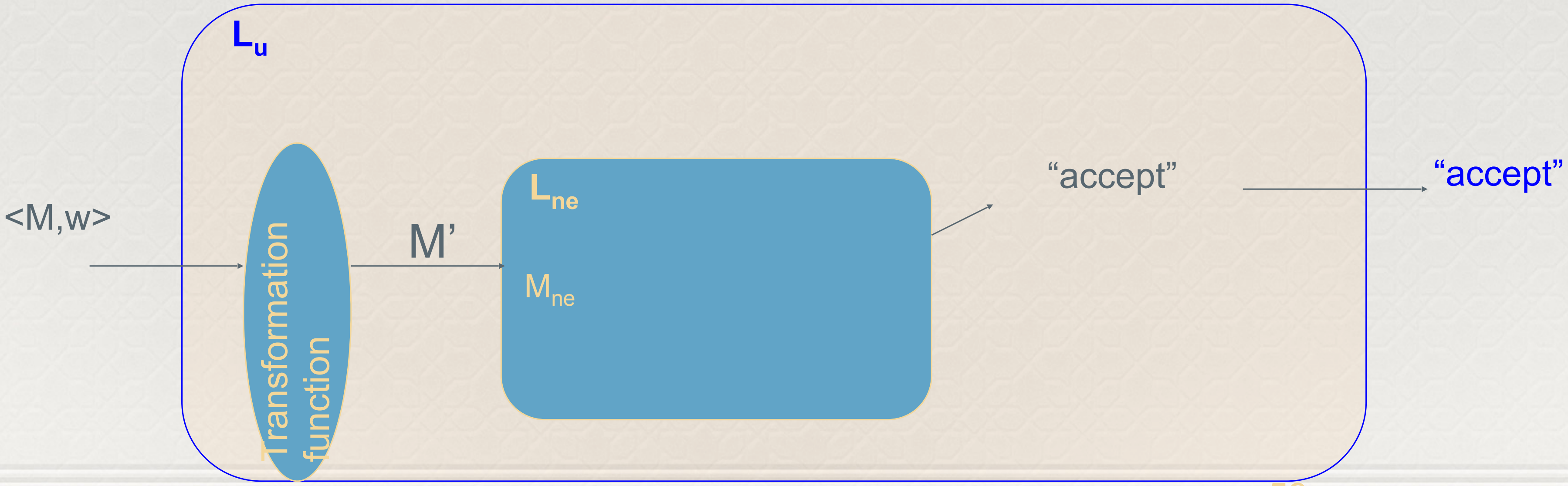
TMs that accept non-empty languages

- $L_{ne} = \{ M \mid L(M) \neq \varnothing \}$
- 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 $L(M') \neq \varnothing$*



Reducability

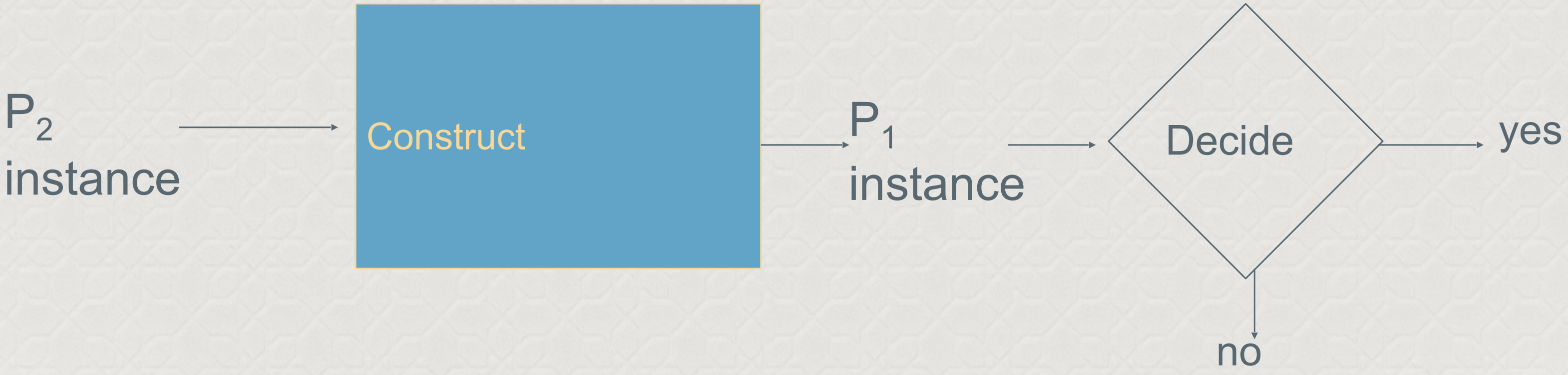
- *To prove:* Problem P_1 is undecidable
- *Given/known:* Problem P_2 is undecidable
- *Reduction idea:*

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
$P_1 \implies P_2$



P_2 instance

Construct

P_1 instance

Decide

yes

no

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*
- *The set of all rational numbers*
- *Reducability*