A Universal Turing Machine

A limitation of Turing Machines:

Turing Machines are "hardwired"

they execute
only one program

Real Computers are re-programmable

Solution: **Universal Turing Machine**

Attributes:

- Reprogrammable machine
- Simulates any other Turing Machine

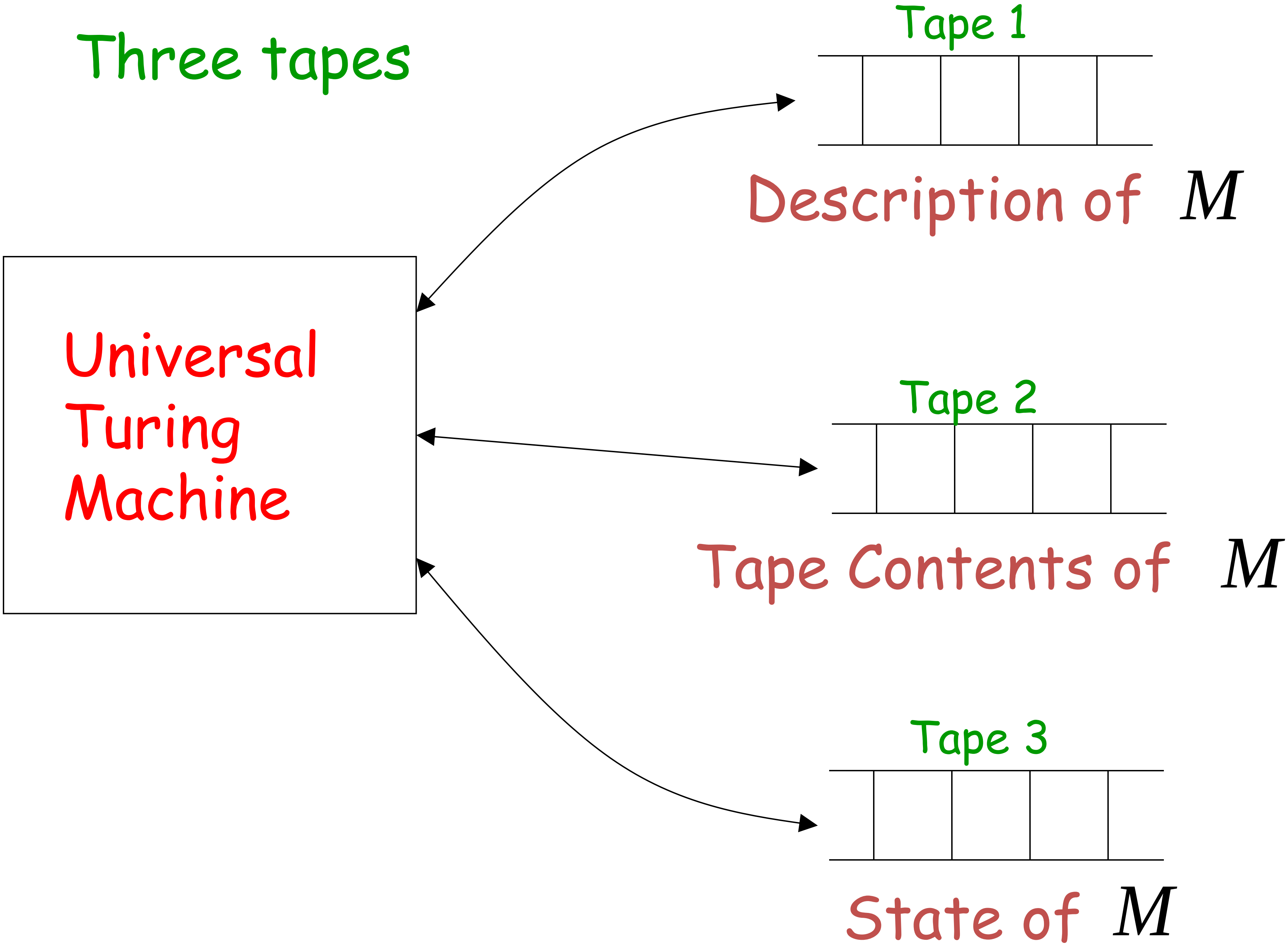
Universal Turing Machine

simulates any Turing Machine M

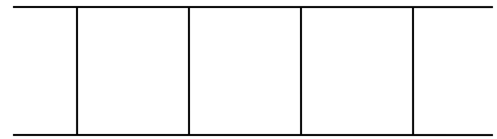
Input of Universal Turing Machine:

Description of transitions of M

Input string of M


Three tapes
Universal Turing Machine
Tape 1
Description of M
Tape 2
Tape Contents of M
Tape 3
State of M

Tape 1



Description of M

We describe Turing machine M
as a string of symbols:

We encode M as a string of symbols

Alphabet Encoding

Symbols:	a	b	c	d	$\cdots$
Encoding:	1	11	111	1111	

State Encoding

States:	q_1	q_2	q_3	q_4	$\cdots$
	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	
Encoding:	1	11	111	1111	

Head Move Encoding

Move:	L	R
	$\downarrow$	$\downarrow$
Encoding:	1	11

Transition Encoding

Transition: $\delta(q_1, a) = (q_2, b, L)$

Encoding: $1\,0\,1\,0\,11\,0\,11\,0\,1$

separator

Turing Machine Encoding

Transitions:

$$\delta(q_1, a) = (q_2, b, L) \qquad \delta(q_2, b) = (q_3, c, R)$$

Encoding:

$$10101101101\ \boxed{00}\ 1101101110111011$$

separator

Tape 1 contents of Universal Turing Machine:

binary encoding
of the simulated machine M

Tape 1

1 0 1 0 11 0 11 0 10011 0 1 10 111 0 111 0 1100…

A Turing Machine is described
with a binary string of 0's and 1's

Therefore:

The set of Turing machines
forms a language:

each string of this language is
the binary encoding of a Turing Machine

Language of Turing Machines

$L = \{$ 010100101, (Turing Machine 1)

00100100101111, (Turing Machine 2)

111010011110010101,

……

…… $\}$

Countable Sets

Infinite sets are either:

Countable

or

Uncountable

Countable set:

There is a one to one correspondence
of
elements of the set
to
Natural numbers (Positive Integers)

very element of the set is mapped to a number
uch that no two elements are mapped to same number)

Example: The set of even integers is countable

Even integers: $0,\ 2,\ 4,\ 6,\ \ldots$
(positive)

Correspondence: $\downarrow\ \downarrow\ \downarrow\ \downarrow$

Positive integers: $1,\ 2,\ 3,\ 4,\ \ldots$

$2n$ corresponds to $n+1$

Example: The set of rational numbers is countable

Rational numbers: $\frac{1}{2}, \frac{3}{4}, \frac{7}{8}, \ldots$

Naïve Approach

Nominator 1

Rational numbers: $\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \ldots$

Correspondence: $\downarrow \quad \downarrow \quad \downarrow$

Positive integers: $1,\ 2,\ 3,\ \ldots$

Doesn't work:

we will never count numbers with nominator 2: $\frac{2}{1}, \frac{2}{2}, \frac{2}{3}, \ldots$

Better Approach

$$\frac{1}{1} \qquad \frac{1}{2} \qquad \frac{1}{3} \qquad \frac{1}{4} \qquad \cdots$$

$$\frac{2}{1} \qquad \frac{2}{2} \qquad \frac{2}{3} \qquad \cdots$$

$$\frac{3}{1} \qquad \frac{3}{2} \qquad \cdots$$

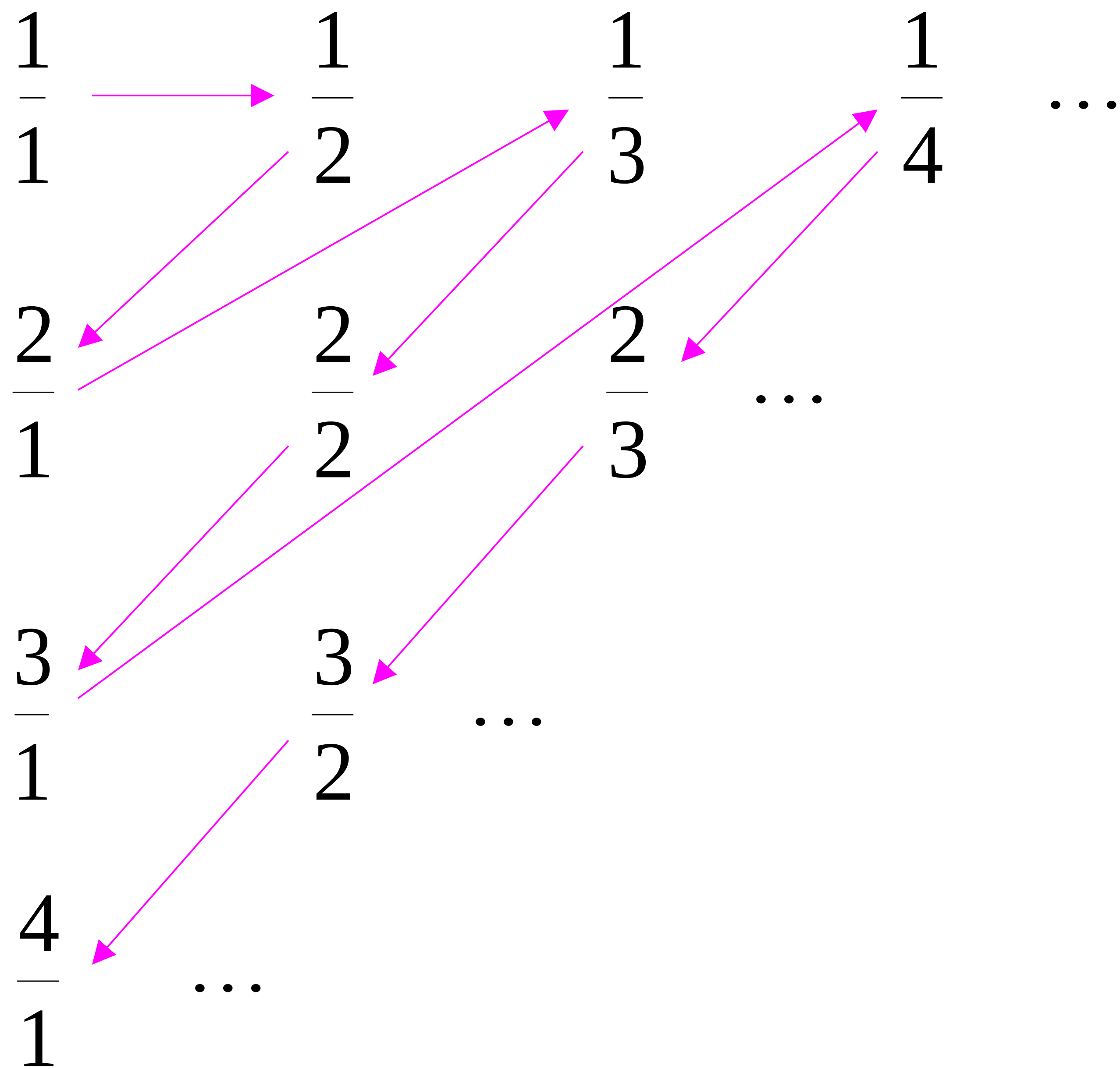
$$\frac{4}{1} \qquad \cdots$$

$$
\begin{array}{lllll}
\frac{1}{1} \longrightarrow & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots \\[2em]
\frac{2}{1} & \frac{2}{2} & \frac{2}{3} & \cdots & \\[2em]
\frac{3}{1} & \frac{3}{2} & \cdots & & \\[2em]
\frac{4}{1} & \cdots & & &
\end{array}
$$

$$\begin{tikzcd}
\frac{1}{1} \arrow[r] & \frac{1}{2} \arrow[dl] & \frac{1}{3} & \frac{1}{4} & \cdots \\
\frac{2}{1} & \frac{2}{2} & \frac{2}{3} & \cdots & \\
\frac{3}{1} & \frac{3}{2} & \cdots & & \\
\frac{4}{1} & \cdots & & &
\end{tikzcd}$$

$$
\begin{array}{cccccc}
\frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots \\[2em]
\frac{2}{1} & \frac{2}{2} & \frac{2}{3} & \cdots & \\[2em]
\frac{3}{1} & \frac{3}{2} & \cdots & & \\[2em]
\frac{4}{1} & \cdots & & &
\end{array}
$$

$$
\begin{array}{ccccc}
\frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots \\[2em]
\frac{2}{1} & \frac{2}{2} & \frac{2}{3} & \cdots & \\[2em]
\frac{3}{1} & \frac{3}{2} & \cdots & & \\[2em]
\frac{4}{1} & \cdots & & &
\end{array}
$$


$\frac{1}{1}$ $\frac{1}{2}$ $\frac{1}{3}$ $\frac{1}{4}$ $\cdots$
$\frac{2}{1}$ $\frac{2}{2}$ $\frac{2}{3}$ $\cdots$
$\frac{3}{1}$ $\frac{3}{2}$ $\cdots$
$\frac{4}{1}$ $\cdots$

Rational Numbers: $\frac{1}{1}, \frac{1}{2}, \frac{2}{1}, \frac{1}{3}, \frac{2}{2}, \dots$

Correspondence:

Positive Integers: $1, \ 2, \ 3, \ 4, \ 5, \ \dots$

We proved:

the set of rational numbers is countable
by describing an enumeration procedure
(enumerator)
for the correspondence to natural
numbers

Definition

Let S be a set of strings (Language)

An **enumerator** for S is a Turing Machine that generates (prints on tape) all the strings of S one by one

and

each string is generated in finite time

strings $s_1, s_2, s_3, \ldots \in S$

Enumerator Machine for S

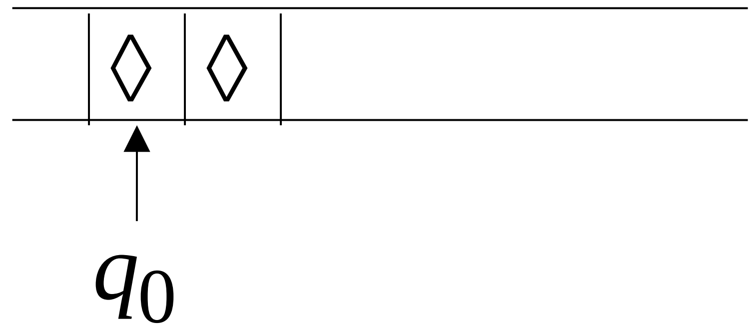
output (on tape) $\longrightarrow$ $s_1, s_2, s_3, \ldots$

Finite time: $t_1, t_2, t_3, \ldots$

Enumerator Machine

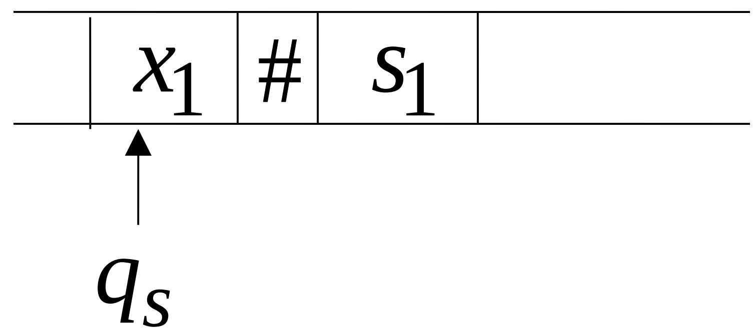
Configuration

Time 0



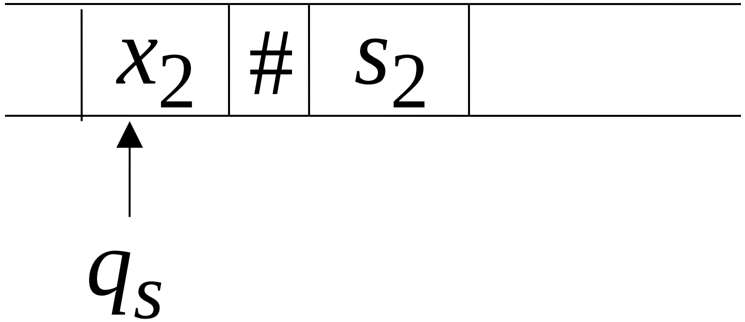
prints s_1

Time t_1



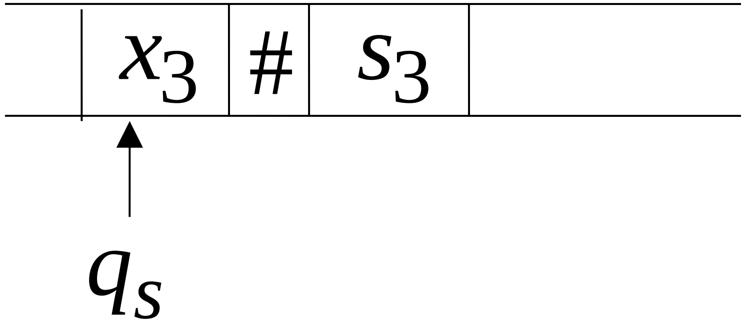
Time t_2

prints s_2



Time t_3

prints s_3



Observation:

If for a set S there is an enumerator, then the set is countable

The enumerator describes the correspondence of S to natural numbers

Example: The set of strings $S = \{a, b, c\}^+$ is countable

Approach:

We will describe an enumerator for S

Naive enumerator:

Produce the strings in lexicographic order:

$$s_1 = a$$

$$s_2 = aa$$

$$\vdots \quad aaa$$

$$aaaa$$

$$\ldots\ldots$$

Doesn't work:

strings starting with b
will never be produced

Better procedure: **Proper Order (Canonical Order)**

1. Produce all strings of length 1
2. Produce all strings of length 2
3. Produce all strings of length 3
4. Produce all strings of length 4

....

Produce strings in
Proper Order:

$$
\left.\begin{array}{l} s_1 = a \\ s_2 = b \\ \vdots \quad c \end{array}\right\} \text{length 1}
$$

$$
\left.\begin{array}{l} aa \\ ab \\ ac \\ ba \\ bb \\ bc \\ ca \\ cb \\ cc \end{array}\right\} \text{length 2}
$$

$$
\left.\begin{array}{l} aaa \\ aab \\ aac \\ \ldots\ldots \end{array}\right\} \text{length 3}
$$

Theorem: The set of all Turing Machines is countable

Proof: Any Turing Machine can be encoded with a binary string of 0's and 1's

Find an enumeration procedure for the set of Turing Machine strings

Enumerator:

Repeat

1. Generate the next binary string of 0's and 1's in proper order

2. Check if the string describes a Turing Machine

   if **YES**: print string on output tape

   if **NO**: ignore string

Binary strings | Turing Machines

0

1

00

01

$\vdots$

1 0 1 0 11 0 11 0 0

1 0 1 0 11 0 11 0 1 $\xrightarrow{s_1}$ 1 0 1 0 11 0 11 0 1

$\vdots$

1 0 11 0 1010010101101 $\xrightarrow{s_2}$ 1 0 11 0 1010010101101

$\vdots$

End of Proof

Uncountable Sets

We will prove that there is a language L' which is not accepted by any Turing machine

Technique:

Turing machines are countable

Languages are uncountable

(there are more languages than Turing Machines)

Definition: A set is uncountable if it is not countable

We will prove that there is a language which is not accepted by any Turing machine

Theorem:

If S is an infinite countable set, then

the powerset 2^S of S is uncountable.

(the powerset 2^S is the set whose elements are all possible sets made from the elements of S)

Proof:

Since S is countable, we can write

$$S = \{s_1, s_2, s_3, \ldots\}$$

Elements of S

Elements of the powerset 2^S have the form:

$$\emptyset$$

$$\{s_1, s_3\}$$

$$\{s_5, s_7, s_9, s_{10}\}$$

......

We encode each element of the powerset with a binary string of 0's and 1's

Powerset element (in arbitrary order)	Binary encoding s_1	s_2	s_3	s_4	$\cdots$
$\{s_1\}$	1	0	0	0	$\cdots$
$\{s_2, s_3\}$	0	1	1	0	$\cdots$
$\{s_1, s_3, s_4\}$	1	0	1	1	$\cdots$

Observation:

Every infinite binary string corresponds to an element of the powerset:

Example: $$1\,0\,0\,1\,1\,1\,0 \quad \cdots$$

Corresponds to: $$\{s_1, s_4, s_5, s_6, \ldots\} \in 2^S$$

Let's assume (for contradiction) that the powerset 2^S is countable

Then: we can enumerate the elements of the powerset

$$2^S = \{t_1, t_2, t_3, \ldots\}$$

Powerset element	suspect that this is the respective **Binary encoding**					
t_1	1	0	0	0	0	$\cdots$
t_2	1	1	0	0	0	$\cdots$
t_3	1	1	0	1	0	$\cdots$
t_4	1	1	0	0	1	$\cdots$
$\cdots$			$\cdots$			

Take the binary string whose bits are the complement of the diagonal

t_1	1	0	0	0	0	$\cdots$
t_2	1	1	0	0	0	$\cdots$
t_3	1	1	0	1	0	$\cdots$
t_4	1	1	0	0	1	$\cdots$

Binary string: $\boxed{t} = 0011\ldots$

(binary complement of diagonal)

The binary string

$$\boldsymbol{t} = 0011\ldots$$

corresponds to an element of the powerset 2^S :

$$\boldsymbol{t} = \{s_3, s_4, \ldots\} \in 2^S$$

Thus, t must be equal to some t_i

$$t = t_i$$

However,

the i-th bit in the encoding of t is the complement of the i-th bit of t_i thus:

$$t \neq t_i$$

Contradiction!!!

Since we have a contradiction:

The powerset 2^S of S is uncountable

End of proof

An Application: Languages

Consider Alphabet : $A = \{a, b\}$

The set of all Strings:

$$S = \{a, b\}^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$$

infinite and countable

(we can enumerate the strings
in proper order)

Consider Alphabet : $A = \{a, b\}$

The set of all Strings:

$$S = \{a, b\}^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$$

infinite and countable

Any language is a subset of S :

$$L = \{aa, ab, aab\}$$

Consider Alphabet : $A = \{a, b\}$

The set of all Strings:

$$S = A^* = \{a, b\}^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$$

infinite and countable

The powerset of S contains all languages:

$$2^S = \{\varnothing, \{\lambda\}, \{a\}, \{a, b\}, \{aa, b\}, \ldots, \underbrace{\{aa, ab, aab\}}_{L}, \ldots\}$$

uncountable

Consider Alphabet : $A = \{a, b\}$

countable

Turing machines: $M_1 \quad M_2 \quad M_3 \quad \cdots$

accepts

Languages accepted
By Turing Machines: $L_1 \quad L_2 \quad L_3 \quad \cdots$

countable

Denote: $X = \{L_1, L_2, L_3, \ldots\}$

countable

Note: $X \subseteq 2^S$

$(S = \{a, b\}^*)$

Languages accepted by Turing machines: X countable

All possible languages: 2^S uncountable

Therefore: $X \neq 2^S$

$$\left(\text{since } X \subseteq 2^S, \text{ we have } X \subset 2^S\right)$$

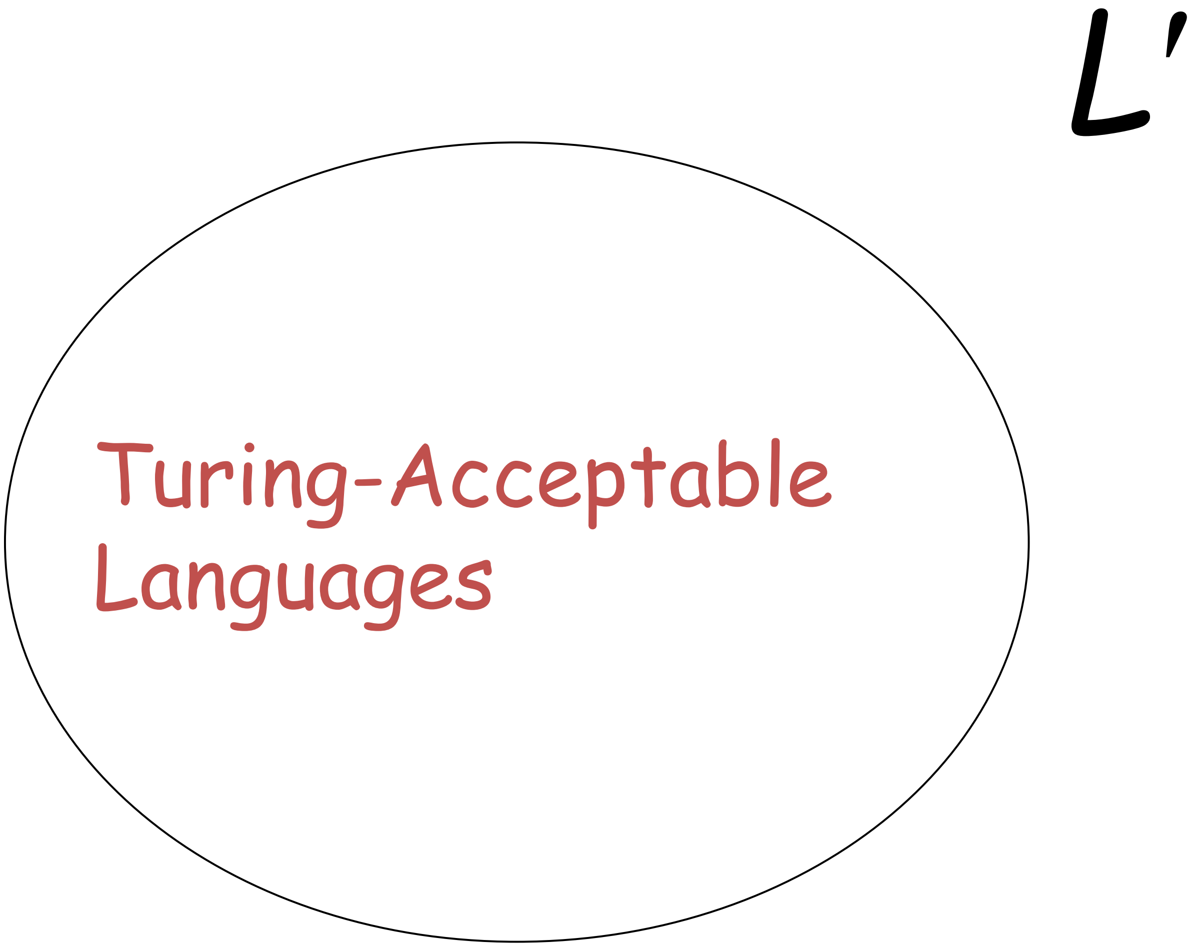
Conclusion:

There is a language L' not accepted by any Turing Machine:

$$\underline{X \subset 2^S} \Longrightarrow \exists L' \in 2^S \text{ and } L' \notin X$$

(Language L' cannot be described by any algorithm)

Non Turing-Acceptable Languages



Note that: $X = \{L_1, L_2, L_3, \ldots\}$

is a *multi-set* (elements may repeat) since a language may be accepted by more than one Turing machine

However, if we remove the repeated elements, the resulting set is again countable since every element still corresponds to a positive integer

Recursively Enumerable and Recursive Languages

Definition:

A language is **recursively enumerable** if some Turing machine accepts it

Let L be a recursively enumerable language

and M the Turing Machine that accepts it

For string w:

if $w \in L$ then M halts in a final state

if $w \notin L$ then M halts in a non-final state

or loops forever

Definition:

A language is **recursive**
if some Turing machine accepts it
and halts on any input string

In other words:

A language is recursive if there is
a membership algorithm for it

Let L be a recursive language

and M the Turing Machine that accepts it

For string w:

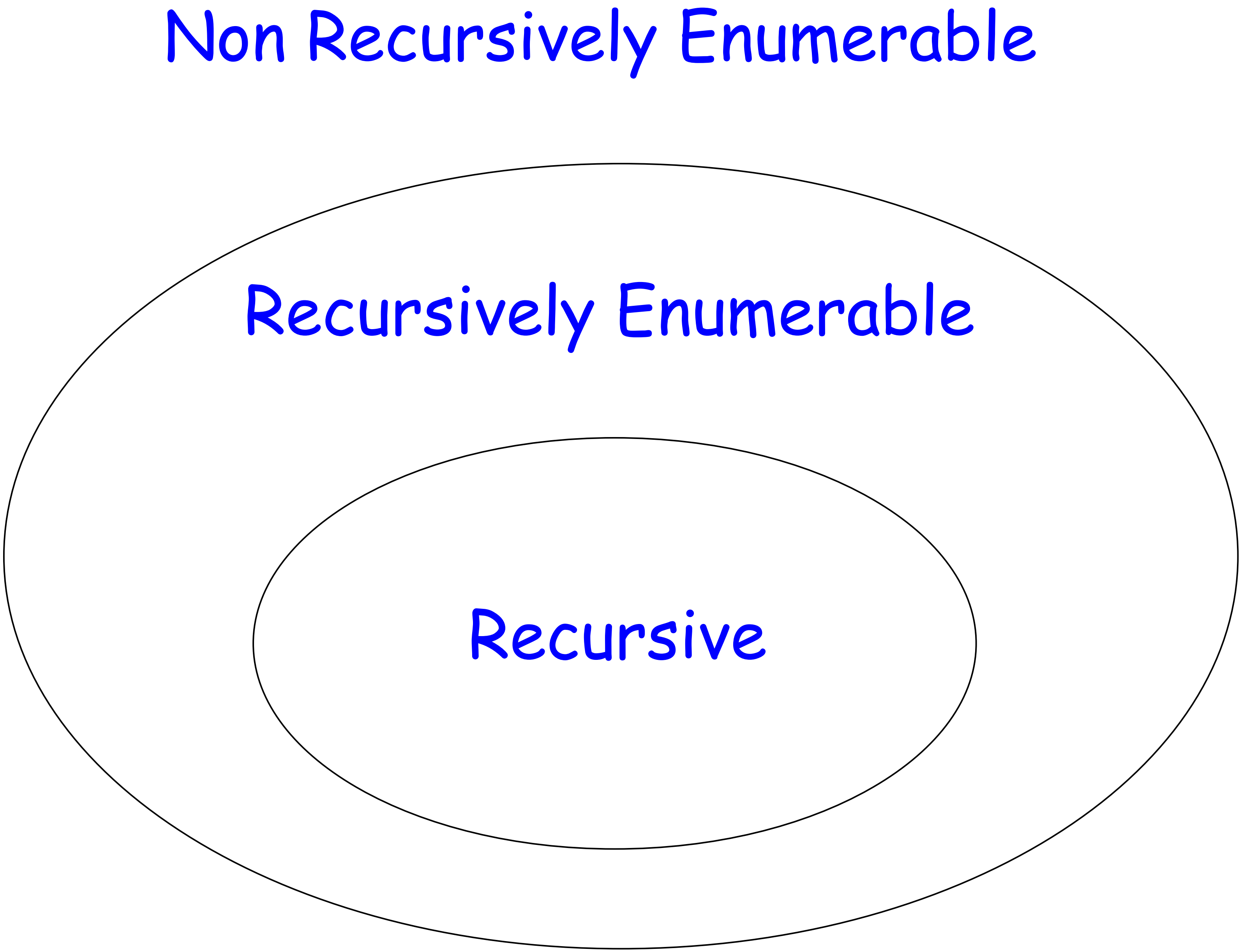
if $w \in L$ then M halts in a final state

if $w \notin L$ then M halts in a non-final state

We will prove:

1. There is a specific language
which is not recursively enumerable
(not accepted by any Turing Machine)

2. There is a specific language
which is recursively enumerable
but not recursive


Non Recursively Enumerable
Recursively Enumerable
Recursive

A Language which
is not
Recursively Enumerable

We want to find a language that is not Recursively Enumerable

This language is not accepted by any Turing Machine

Consider alphabet $\{a\}$

Strings: $a, \; aa, \; aaa, \; aaaa, \; \ldots$

$a^1 \quad a^2 \quad a^3 \quad a^4 \quad \ldots$

Consider Turing Machines
that accept languages over alphabet $\{a\}$

They are countable:

$$M_1,\ M_2,\ M_3,\ M_4,\ \ldots$$

Example language accepted by M_i

$$L(M_i) = \{aa, aaaa, aaaaaa\}$$

$$L(M_i) = \{a^2, a^4, a^6\}$$

Alternative representation

	a^1	a^2	a^3	a^4	a^5	a^6	a^7	$\cdots$
$L(M_i)$	0	1	0	1	0	1	0	$\cdots$

	a^1	a^2	a^3	a^4	$\cdots$
$L(M_1)$	0	1	0	1	$\cdots$
$L(M_2)$	1	0	0	1	$\cdots$
$L(M_3)$	0	1	1	1	$\cdots$
$L(M_4)$	0	0	0	1	$\cdots$

Consider the language

$$L = \{a^i : a^i \in L(M_i)\}$$

L consists from the 1's in the diagonal

	a^1	a^2	a^3	a^4	$\cdots$
$L(M_1)$	0	1	0	1	$\cdots$
$L(M_2)$	1	0	0	1	$\cdots$
$L(M_3)$	0	1	1	1	$\cdots$
$L(M_4)$	0	0	0	1	$\cdots$

$$L = \{a^3, a^4, \ldots\}$$

Consider the language $\overline{L}$

$$L = \{a^i : a^i \in L(M_i)\}$$

$$\overline{L} = \{a^i : a^i \notin L(M_i)\}$$

$\overline{L}$ consists of the 0's in the diagonal

	a^1	a^2	a^3	a^4	$\cdots$
$L(M_1)$	0	1	0	1	$\cdots$
$L(M_2)$	1	0	0	1	$\cdots$
$L(M_3)$	0	1	1	1	$\cdots$
$L(M_4)$	0	0	0	1	$\cdots$

$$\overline{L} = \{a^1, a^2, \ldots\}$$

Theorem:

Language $\overline{L}$ is not recursively enumerable

Proof:

Assume for contradiction that

$\overline{L}$ is recursively enumerable

There must exist some machine M_k that accepts $\overline{L}$

$$L(M_k) = \overline{L}$$

	a^1	a^2	a^3	a^4	$\cdots$
$L(M_1)$	0	1	0	1	$\cdots$
$L(M_2)$	1	0	0	1	$\cdots$
$L(M_3)$	0	1	1	1	$\cdots$
$L(M_4)$	0	0	0	1	$\cdots$

Question: $M_k = M_1$?

	a^1	a^2	a^3	a^4	$\cdots$
$L(M_1)$	0	1	0	1	$\cdots$
$L(M_2)$	1	0	0	1	$\cdots$
$L(M_3)$	0	1	1	1	$\cdots$
$L(M_4)$	0	0	0	1	$\cdots$

$$a^1 \in L(M_k)$$

Answer: $M_k \neq M_1$

$$a^1 \notin L(M_1)$$

	a^1	a^2	a^3	a^4	$\cdots$
$L(M_1)$	0	1	0	1	$\cdots$
$L(M_2)$	1	0	0	1	$\cdots$
$L(M_3)$	0	1	1	1	$\cdots$
$L(M_4)$	0	0	0	1	$\cdots$

Question: $M_k = M_2$?

	a^1	a^2	a^3	a^4	$\cdots$
$L(M_1)$	0	1	0	1	$\cdots$
$L(M_2)$	1	0	0	1	$\cdots$
$L(M_3)$	0	1	1	1	$\cdots$
$L(M_4)$	0	0	0	1	$\cdots$

$$a^2 \in L(M_k)$$

Answer: $M_k \neq M_2$

$$a^2 \notin L(M_2)$$

	a^1	a^2	a^3	a^4	$\cdots$
$L(M_1)$	0	1	0	1	$\cdots$
$L(M_2)$	1	0	0	1	$\cdots$
$L(M_3)$	0	1	1	1	$\cdots$
$L(M_4)$	0	0	0	1	$\cdots$

Question: $M_k = M_3$?

	a^1	a^2	a^3	a^4	$\cdots$
$L(M_1)$	0	1	0	1	$\cdots$
$L(M_2)$	1	0	0	1	$\cdots$
$L(M_3)$	0	1	1	1	$\cdots$
$L(M_4)$	0	0	0	1	$\cdots$

$$a^3 \notin L(M_k)$$

Answer: $M_k \neq M_3$

$$a^3 \in L(M_3)$$

Similarly: $M_k \neq M_i$ for any i

Because either:

$$a^i \in L(M_k) \quad \text{or} \quad a^i \notin L(M_k)$$

$$a^i \notin L(M_i) \qquad\qquad a^i \in L(M_i)$$

Therefore, the machine M_k cannot exist

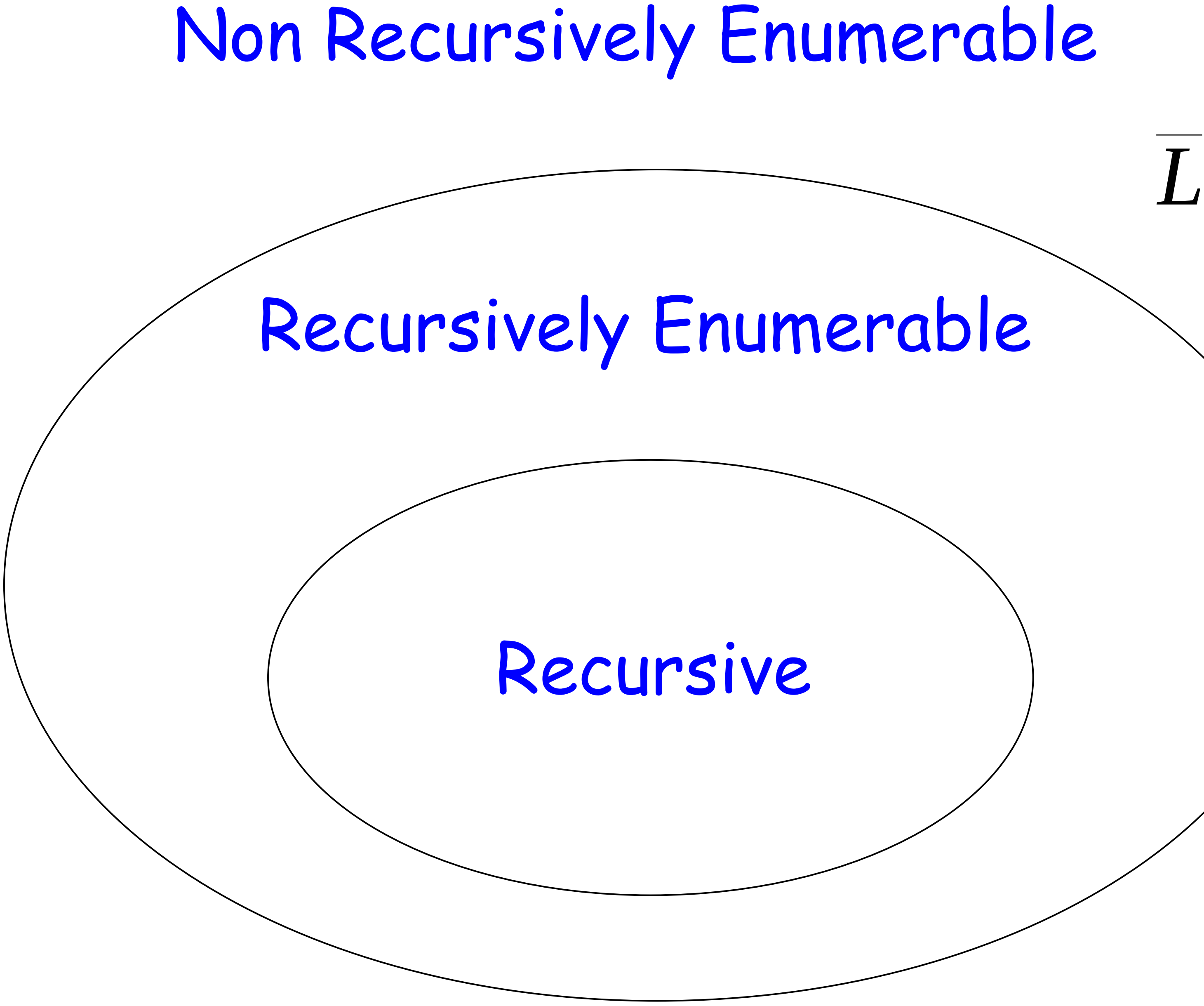
Therefore, the language $\overline{L}$
is not recursively enumerable

End of Proof

Observation:

There is no algorithm that describes $\overline{L}$

(otherwise $\overline{L}$ would be accepted by some Turing Machine)


Non Recursively Enumerable
$\overline{L}$
Recursively Enumerable
Recursive

A Language which is Recursively Enumerable and not Recursive

We want to find a language which

Is recursively enumerable

But not recursive

There is a Turing Machine that accepts the language

The machine doesn't halt on some input

We will prove that the language

$$L = \{a^i : a^i \in L(M_i)\}$$

Is recursively enumerable
but not recursive

	a^1	a^2	a^3	a^4	$\cdots$
$L(M_1)$	0	1	0	1	$\cdots$
$L(M_2)$	1	0	0	1	$\cdots$
$L(M_3)$	0	1	1	1	$\cdots$
$L(M_4)$	0	0	0	1	$\cdots$

$$L = \{a^3, a^4, \ldots\}$$

Theorem:

The language $L = \{a^i : a^i \in L(M_i)\}$

is recursively enumerable

Proof:

We will give a Turing Machine that accepts L

Turing Machine that accepts L

For any input string w

- Compute i, for which $w = a^i$
- Find Turing machine M_i

  (using an enumeration procedure for Turing Machines)

- Simulate M_i on input a^i
- If M_i accepts, then accept w

End of Proof

Observation:

Recursively enumerable

$$L = \{a^i : a^i \in L(M_i)\}$$

Not recursively enumerable

$$\overline{L} = \{a^i : a^i \notin L(M_i)\}$$

(Thus, also not recursive)

Theorem:

The language $L = \{a^i : a^i \in L(M_i)\}$

is not recursive

Proof:

Assume for contradiction that L is recursive

Then $\overline{L}$ is recursive:

Take the Turing Machine M that accepts L

M halts on any input:

If M accepts then reject
If M rejects then accept

Therefore:

$\overline{L}$ is recursive

But we know:

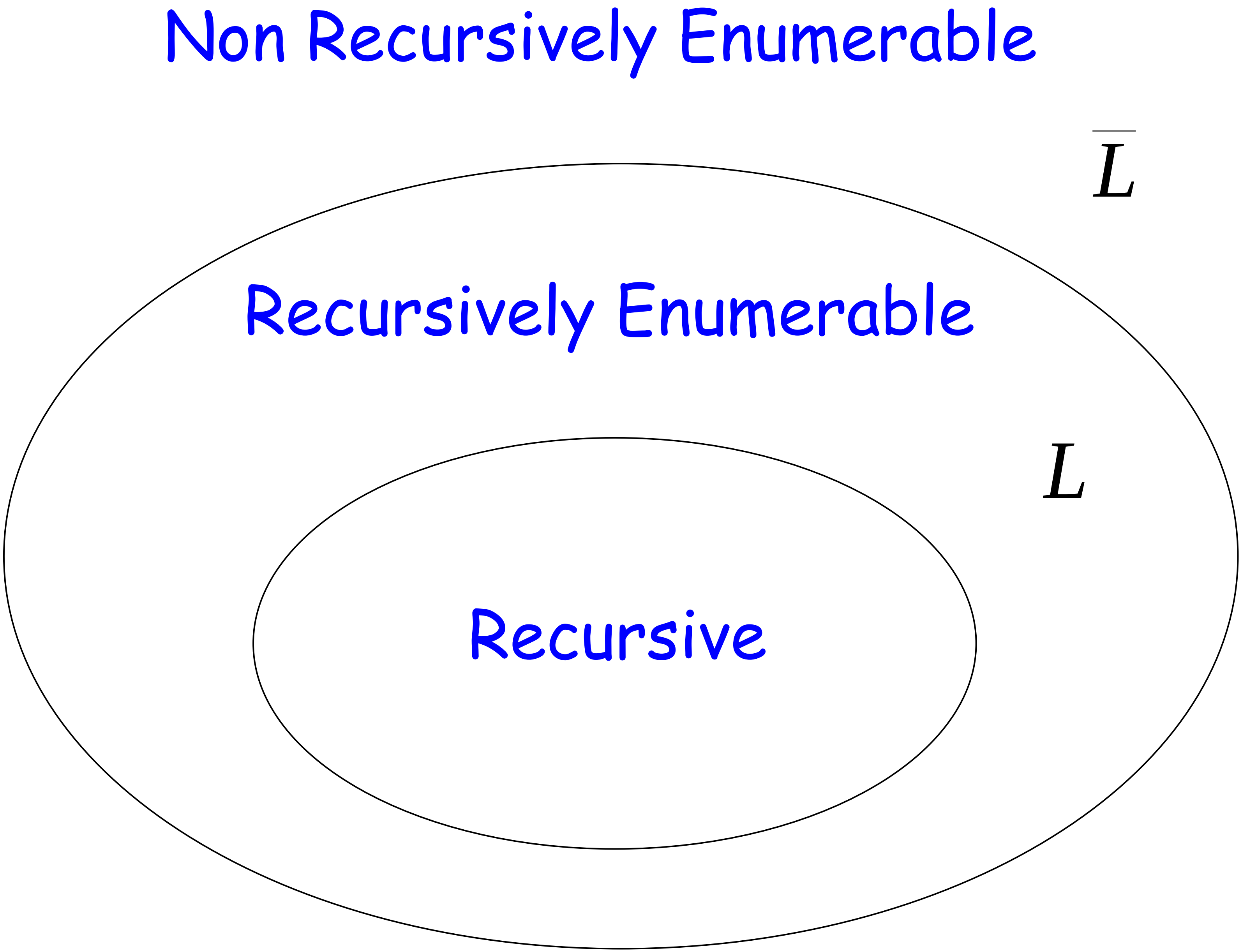
$\overline{L}$ is not recursively enumerable

thus, not recursive

CONTRADICTION!!!!

Therefore, L is not recursive

End of Proof


Non Recursively Enumerable
$\overline{L}$
Recursively Enumerable
L
Recursive

Turing acceptable languages
and
Enumeration Procedures

We will prove:

(weak result)

- If a language is recursive then there is an enumeration procedure for it

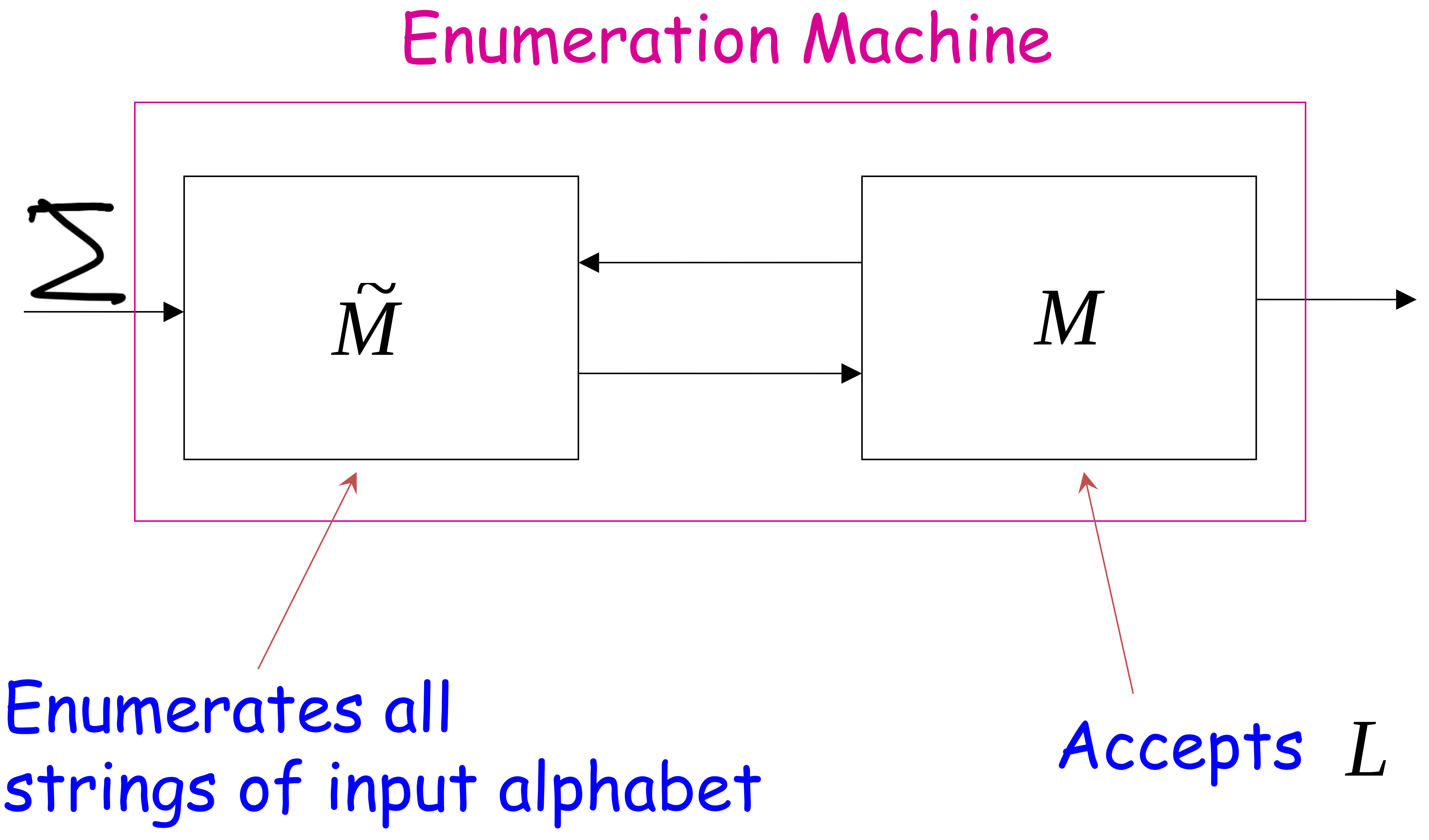
(strong result)

- A language is recursively enumerable if and only if there is an enumeration procedure for it

Theorem:

if a language L is recursive then there is an enumeration procedure for it

Proof:



Enumeration Machine
Σ^*
$\tilde{M}$
M
Enumerates all strings of input alphabet
Accepts L

If the alphabet is $\{a, b\}$ then $\tilde{M}$ can enumerate strings as follows:

$$a$$
$$b$$
$$aa$$
$$ab$$
$$ba$$
$$bb$$
$$aaa$$
$$aab$$
$$\ldots\ldots$$

Enumeration procedure

Repeat:

$\tilde{M}$ generates a string w

M checks if $w \in L$

YES: print w to output

NO: ignore w

End of Proof

Example: $L = \{b, ab, bb, aaa, ...\}$

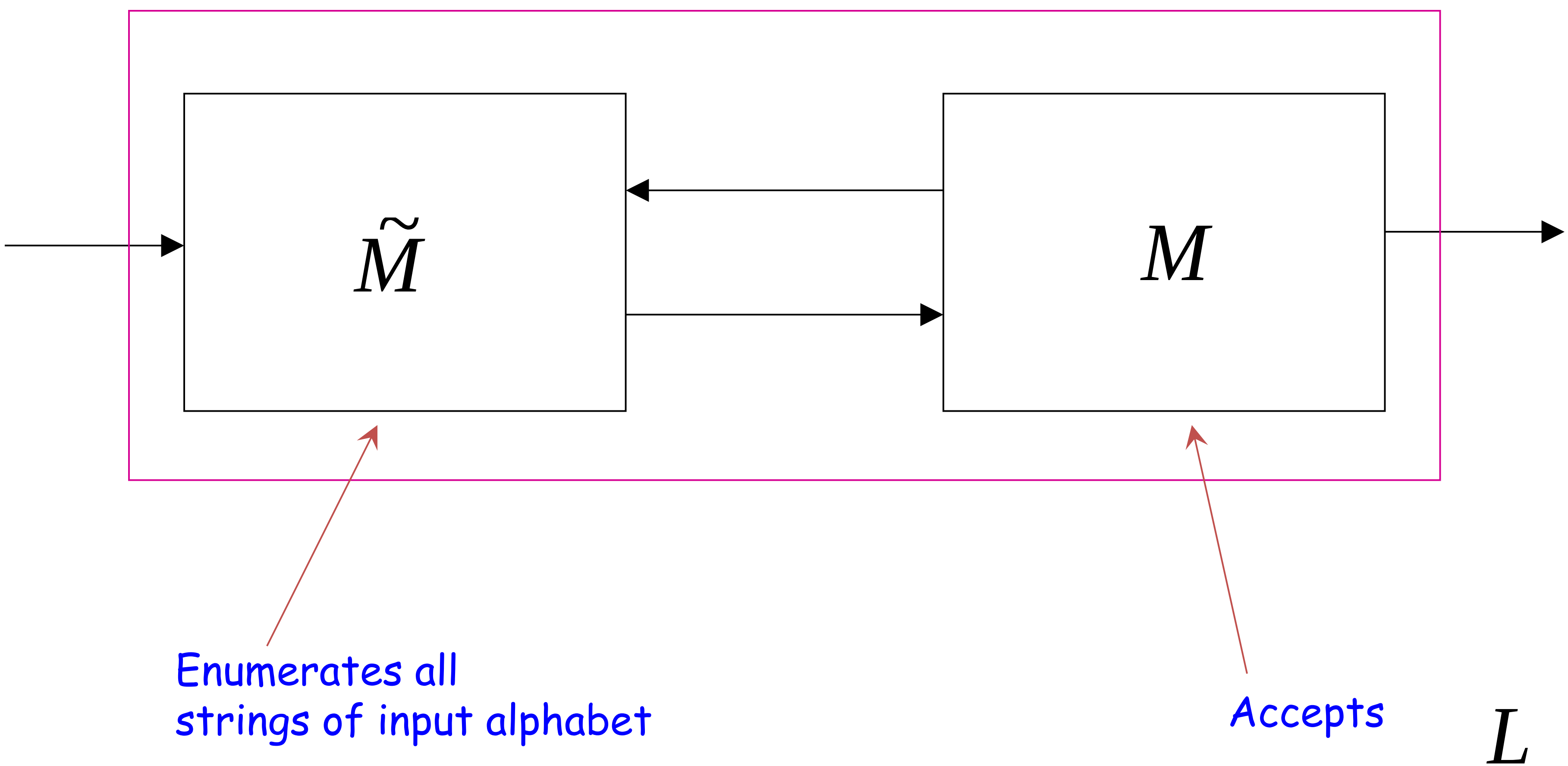
$\tilde{M}$	$L(M)$	Enumeration Output
a		
b	b	b
aa		
ab	ab	ab
ba		
bb	bb	bb
aaa	aaa	aaa
aab		
......	......	......

Theorem:

if language L is recursively enumerable then there is an enumeration procedure for it

Proof:

Enumeration Machine



If the alphabet is $\{a, b\}$ then $\tilde{M}$ can enumerate strings as follows:

$$a$$
$$b$$
$$aa$$
$$ab$$
$$ba$$
$$bb$$
$$aaa$$
$$aab$$

NAIVE APPROACH

Enumeration procedure

Repeat: $\tilde{M}$ generates a string w

M checks if $w \in L$

YES: print w to output

NO: ignore w

Problem: If $w \notin L$
machine M may loop forever

BETTER APPROACH

$\tilde{M}$ Generates first string w_1

M executes first step on w_1

$\tilde{M}$ Generates second string w_2

M executes first step on w_2

second step on w_1

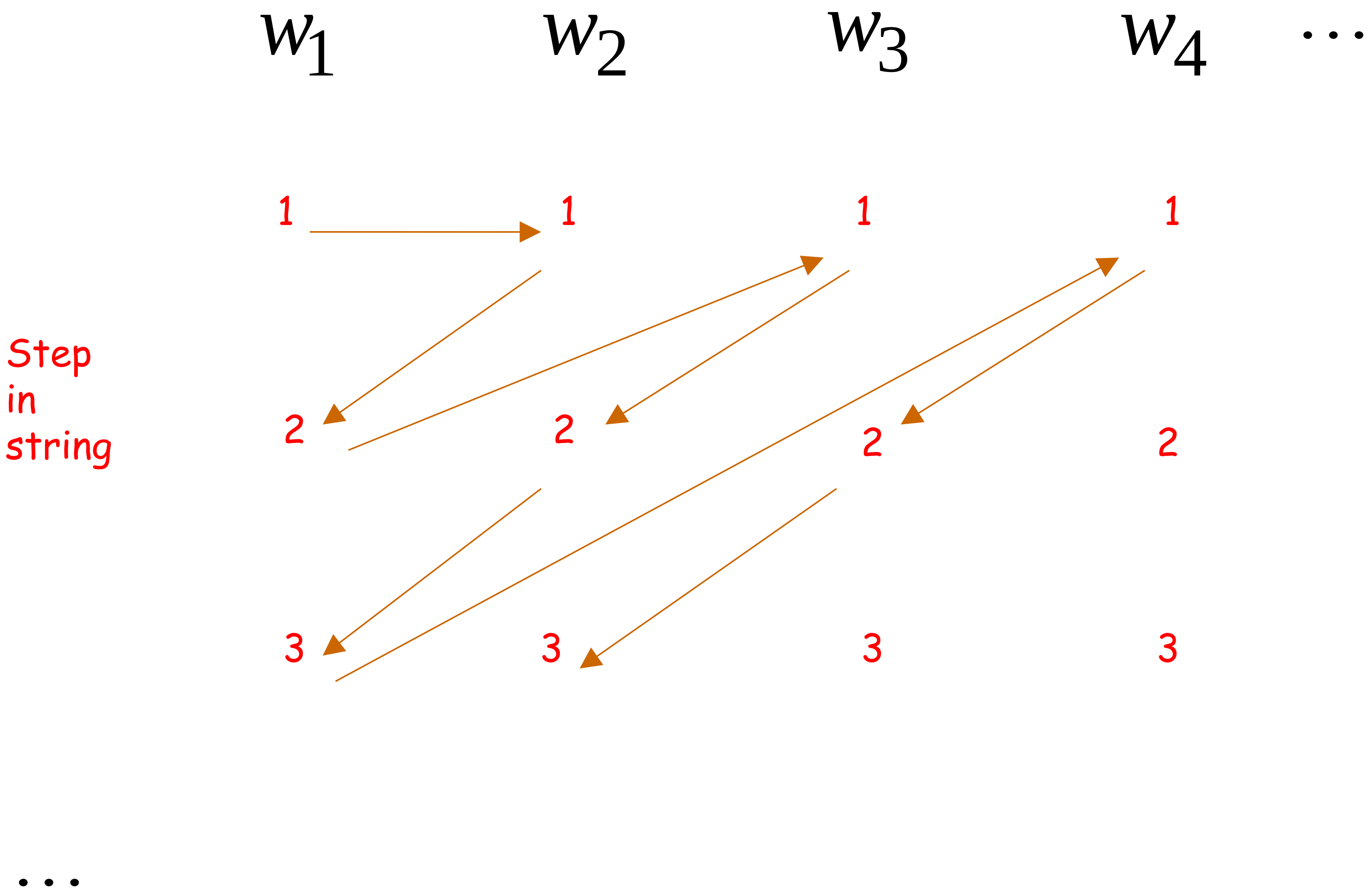
$\tilde{M}$ Generates third string w_3

M executes first step on w_3

second step on w_2

third step on w_1

And so on............



w_1
w_2
w_3
w_4
...
1
1
1
1
Step in string
2
2
2
2
3
3
3
3
...

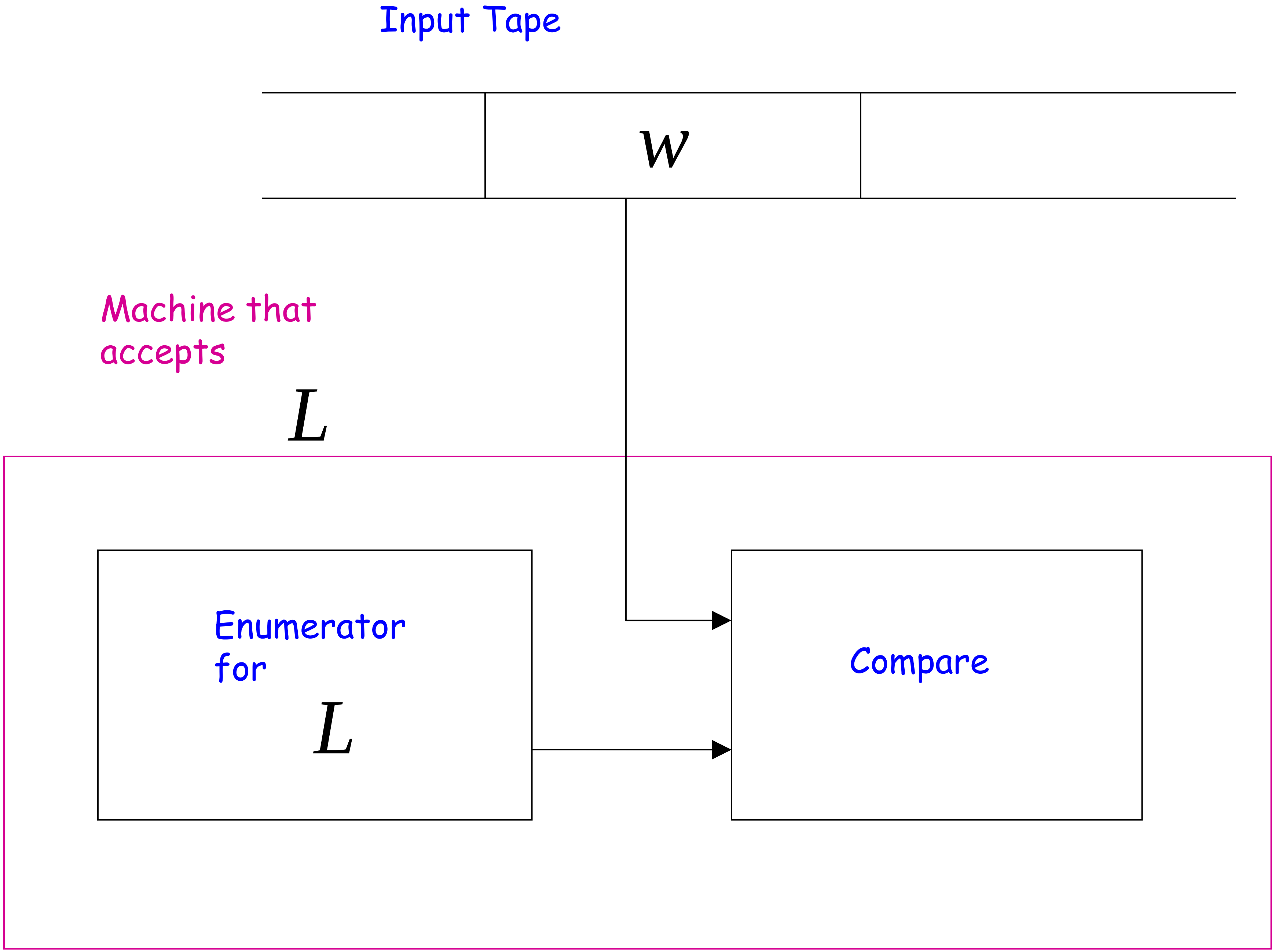
If for any string w_i machine M halts in a final state then it prints w_i on the output

End of Proof

Theorem:

If for language L
there is an enumeration procedure
then L is recursively enumerable

Proof:



Input Tape
w
Machine that accepts
L
Enumerator for
L
Compare

Turing machine that accepts L

For input string w

Repeat:

- Using the enumerator, generate the next string of L

- Compare generated string with w

  If same, accept and exit loop

End of Proof

We have proven:

A language is recursively enumerable
if and only if
there is an enumeration procedure for it