Turing Machines and Chomsky Hierarchy

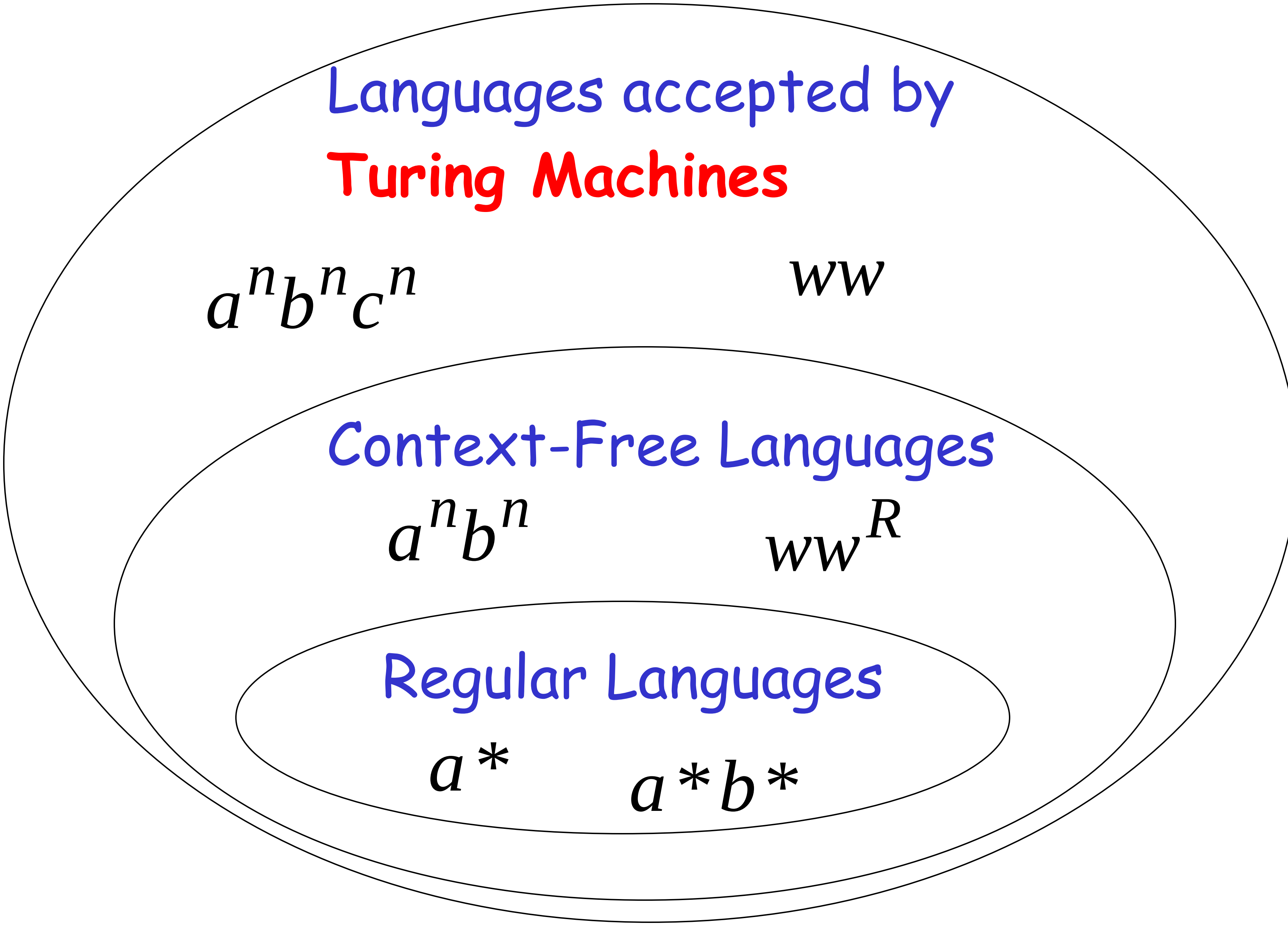
Formal Languages and Abstract Machines

Week 11

Baris E. Suzek, PhD

Outline

- Review of last week
- Universal Turing Machine
- Countable/uncountable Sets
- Linear Bounded Automata
- Chomsky Hierarchy and Recursively Enumerable Languages


Languages accepted by
Turing Machines
$a^n b^n c^n$
ww
Context-Free Languages
$a^n b^n$
ww^R
Regular Languages
a^*
$a^* b^*$

Acceptance

Accept Input → If machine halts in a final state

Reject Input → If machine halts in a non-final state

or

If machine enters an *infinite loop*

Standard Turing Machine

The machine we described is the standard:

- Deterministic
- Infinite tape in both directions
- Tape is the input/output file

Turing Machine:

Input alphabet

Tape alphabet

States

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

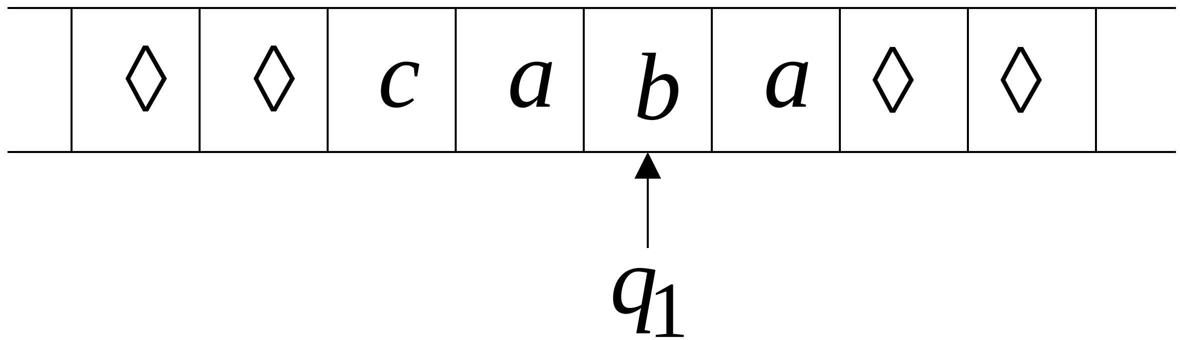
Transition function

Final states

Initial state

blank

Configuration



Instantaneous description: $ca\ q_1\ ba$

The Accepted Language

For any Turing Machine M

$$L(M) = \{w : \quad q_0\, w \overset{*}{\succ} x_1\, q_f\, x_2\}$$

Initial state

Final state

In other words:

A function f is computable if
there is a Turing Machine M such that:

$$q_0 \, w \overset{*}{\succ} q_f \, f(w)$$

Initial Configuration

Final Configuration

For all $w \in D$ Domain

Example

The function $f(x, y) = x + y$ is computable

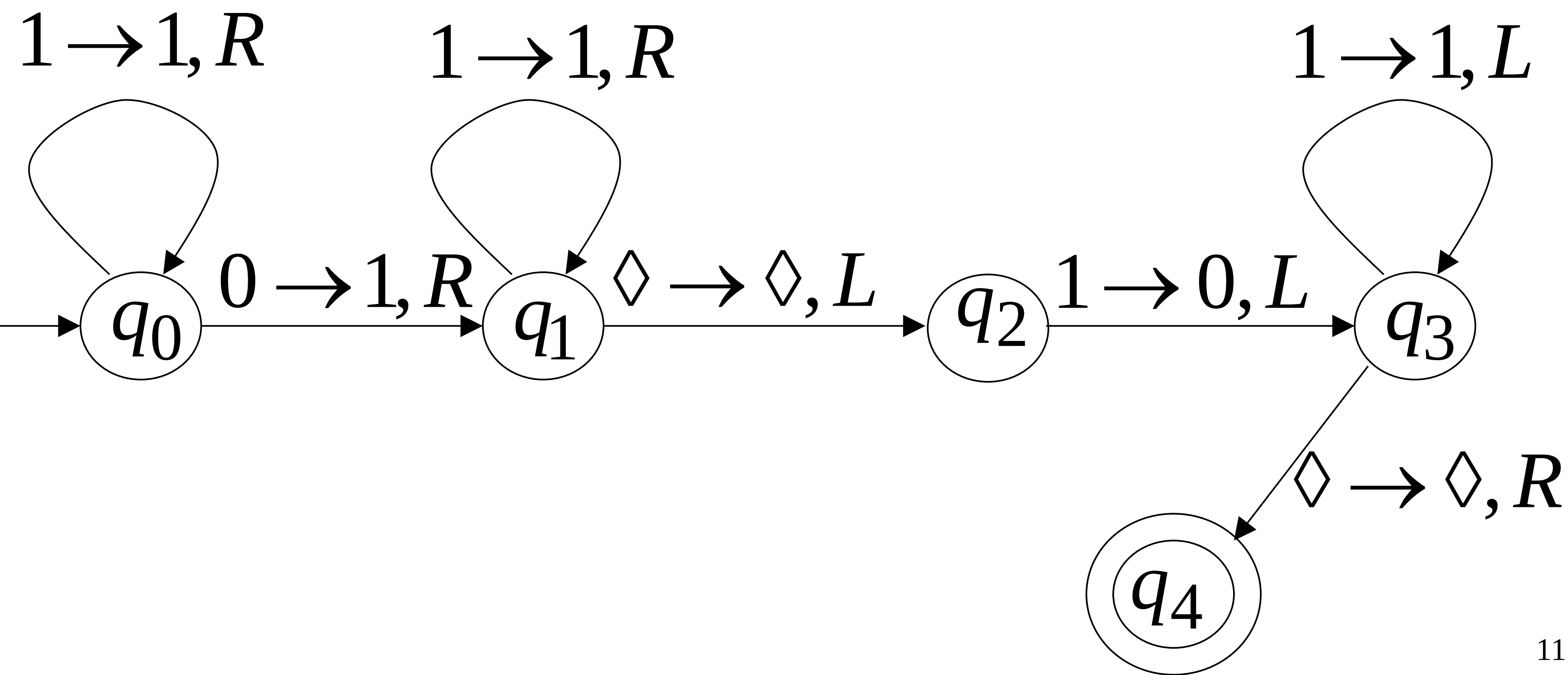
x, y are integers

Turing Machine:

Input string: $x0y$ unary

Output string: $xy0$ unary

Turing machine for function $f(x,y) = x + y$



Turing Machine Pseudocode for $f(x) = 2x$

- Replace every 1 with \$

- Repeat:
  - Find rightmost \$, replace it with 1

  - Go to right end, insert 1

Until no more \$ remain

Definition of Algorithm:

An algorithm for function $f(w)$

is a

Turing Machine which computes $f(w)$

Algorithms are Turing Machines

When we say:

There exists an algorithm

We mean:

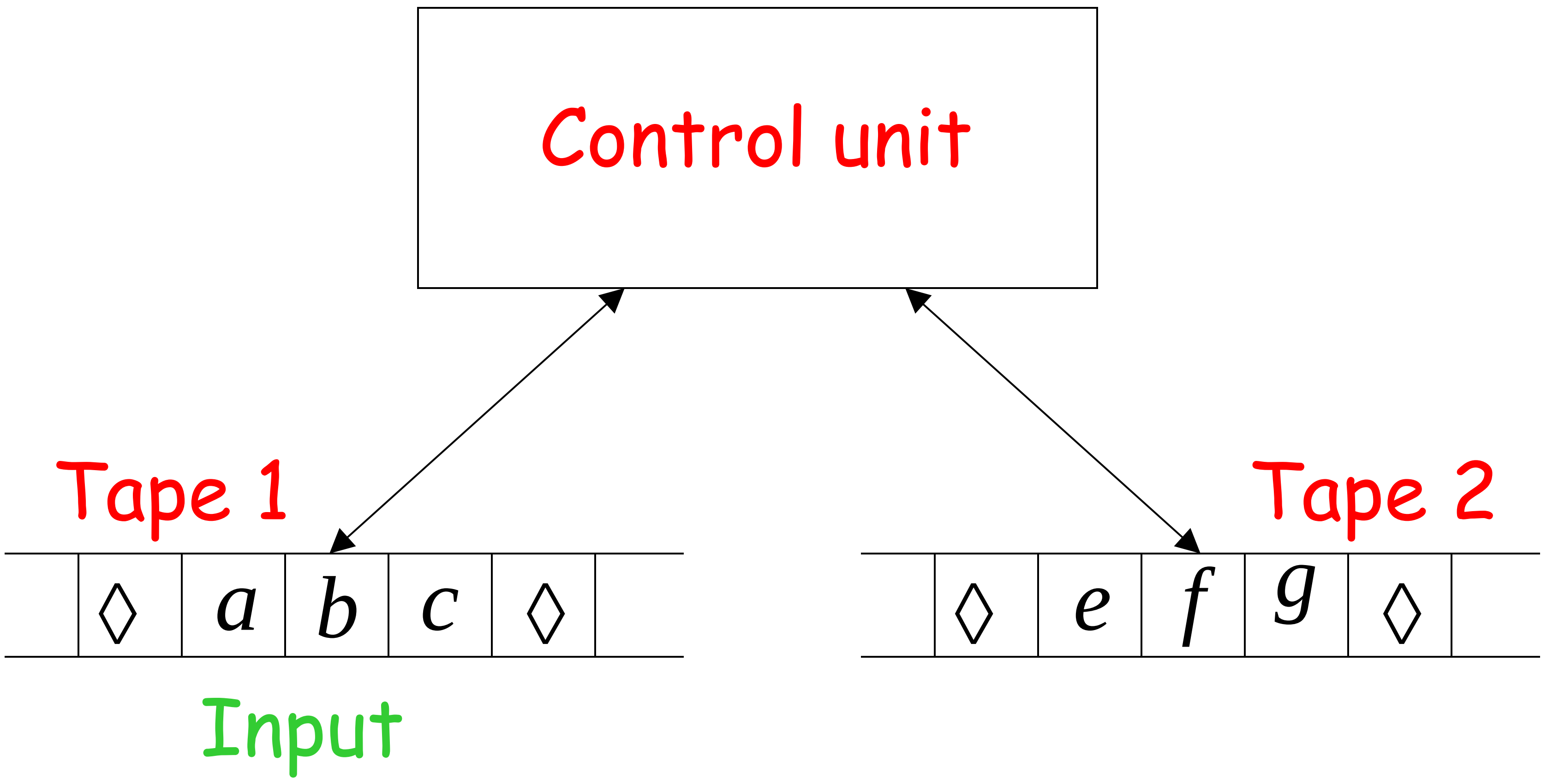
There exists a Turing Machine
that executes the algorithm

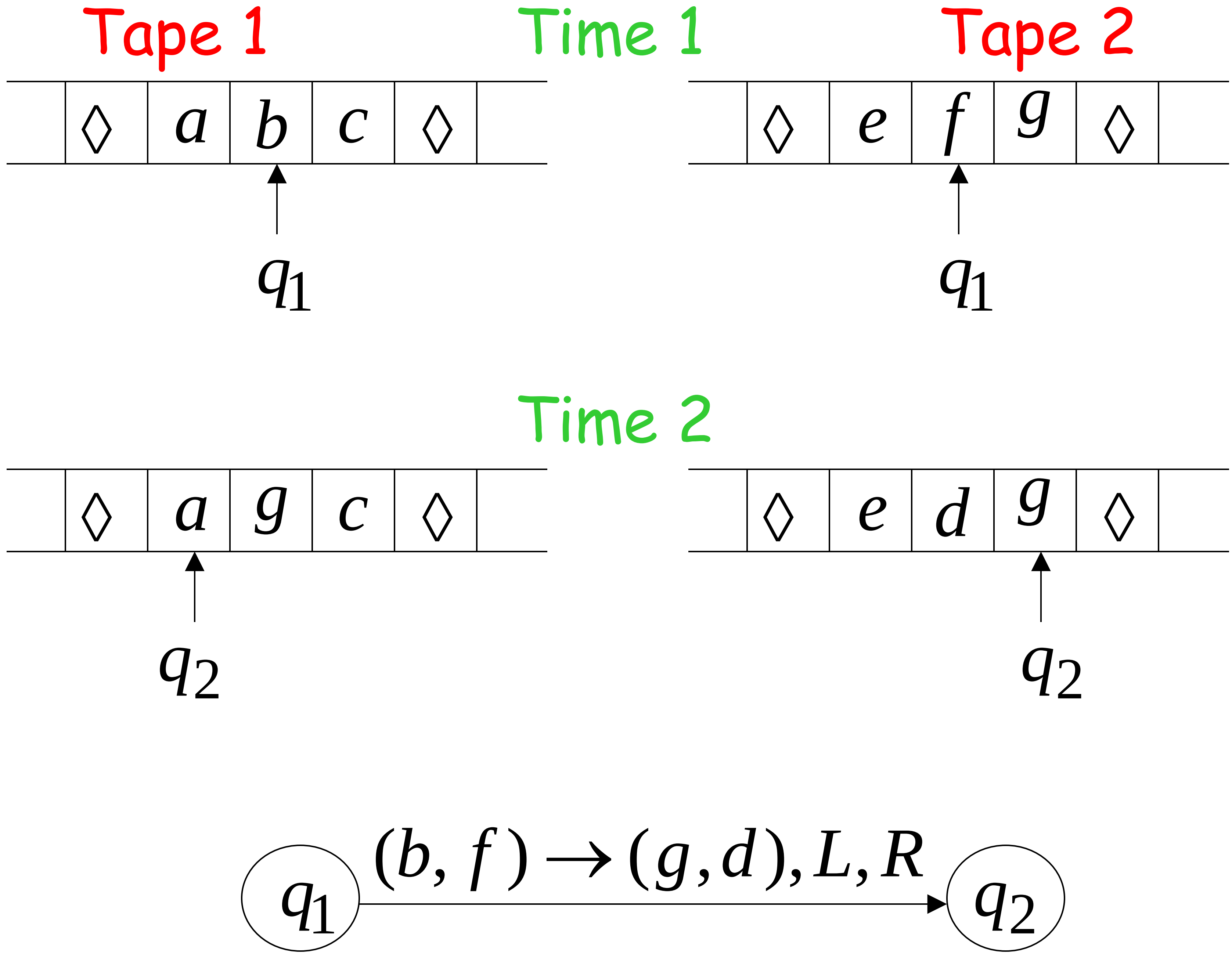
Variations of the Standard Model

Turing machines with:

- Stay-Option
- Semi-Infinite Tape
- Off-Line
- Multitape
- Multidimensional

Multitape Turing Machines




Tape 1
Time 1
Tape 2
◊ a b c ◊
◊ e f g ◊
q_1
q_1
Time 2
◊ a g c ◊
◊ e d g ◊
q_2
q_2
q_1
$(b, f) \rightarrow (g, d), L, R$
q_2

Outline

- Review of last week
- Universal Turing Machine
- Countable/uncountable Sets
- Linear Bounded Automata
- Chomsky Hierarchy and Recursively Enumerable Languages

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

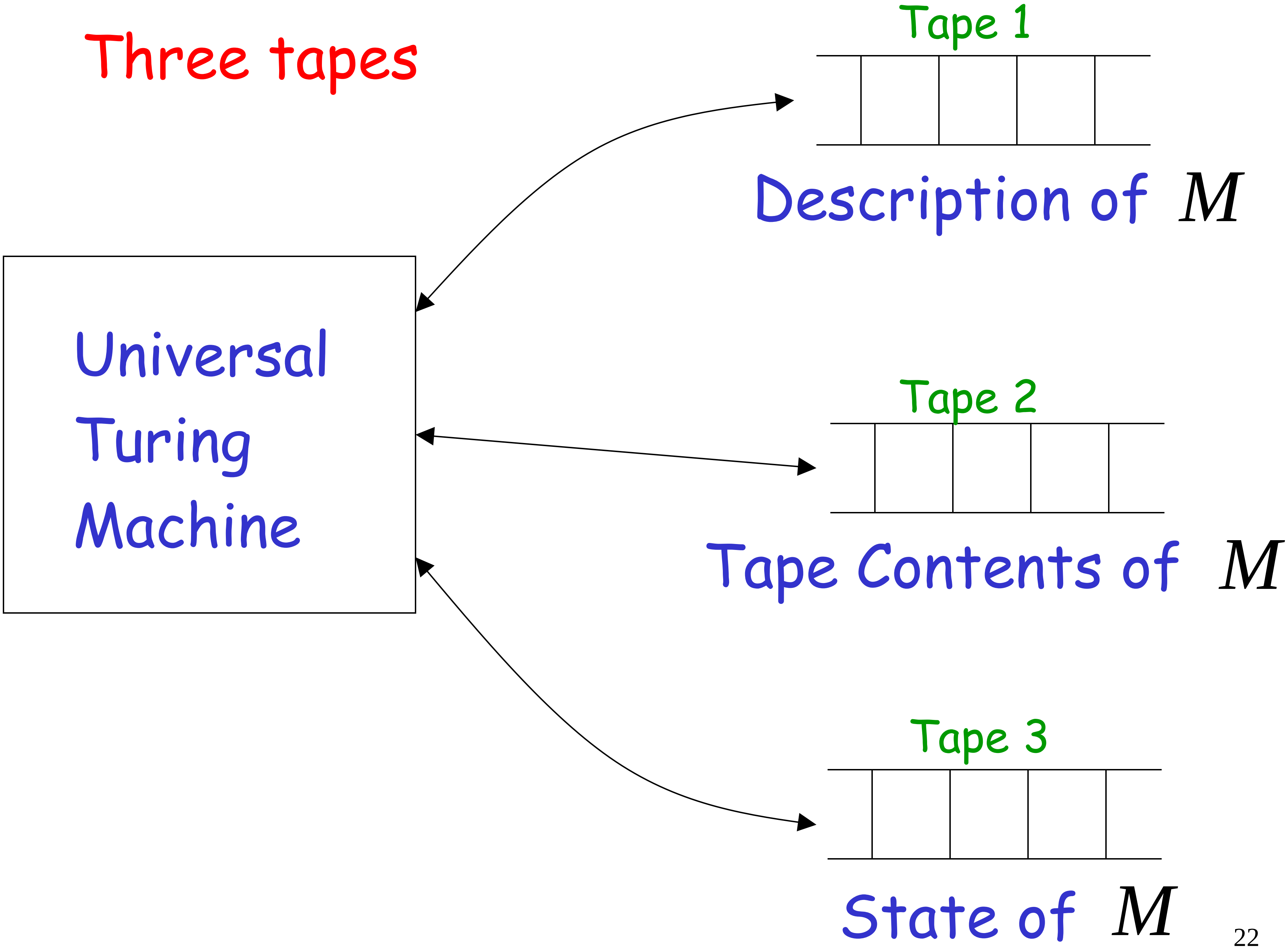
simulates any other Turing Machine M

Input of Universal Turing Machine:

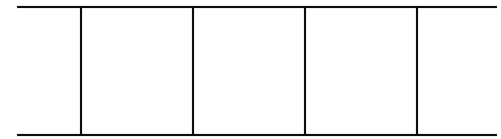
Description of transitions of M

Initial tape contents 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	$\cdots$
	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	
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

Machine Encoding

Transitions:

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

Encoding:

$$10101101101 \; 00 \; 1101101110111011$$

separator

Tape 1 contents of Universal Turing Machine:

encoding of the simulated machine M
as a binary string of 0's and 1's

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 the language is the binary encoding of a Turing Machine

Language of Turing Machines

L = { 010100101, (Turing Machine 1)

00100100101111, (Turing Machine 2)

111010011110010101, ……

…… }

Outline

- Review of last week
- Universal Turing Machine
- Countable/uncountable Sets
- Linear Bounded Automata
- Chomsky Hierarchy and Recursively Enumerable Languages

Infinite sets are either:

Countable

or

Uncountable

Countable set:

Any finite set

or

Any *Countably infinite* set:

There is a one to one correspondence

between

elements of the set

and

Natural numbers

Example: The set of even integers is countable

Even integers: $0, \ 2, \ 4, \ 6, \ \ldots$

Correspondence:

$$\begin{array}{ccccc} 0, & 2, & 4, & 6, & \ldots \\ \downarrow & \downarrow & \downarrow & \downarrow & \\ 1, & 2, & 3, & 4, & \ldots \end{array}$$

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 Proof

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

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

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

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

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

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

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

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

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

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

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

Correspondence:

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

We proved:

the set of rational numbers is countable by describing an enumeration procedure

Definition

Let S be a set of strings

An **enumeration procedure** for S is a Turing Machine that generates all strings of S one by one

and

each string is generated in finite time

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

Enumeration Machine for S

output (on tape)

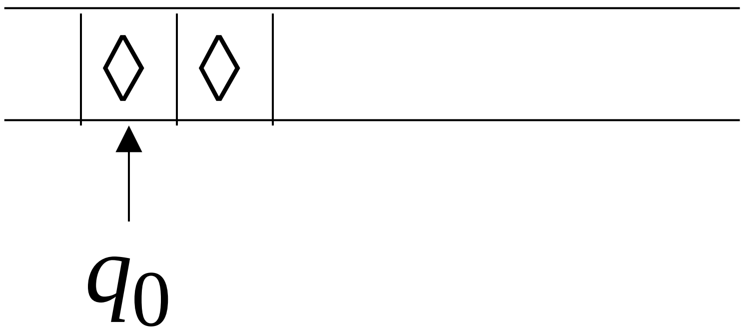
$s_1, s_2, s_3, \ldots$

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

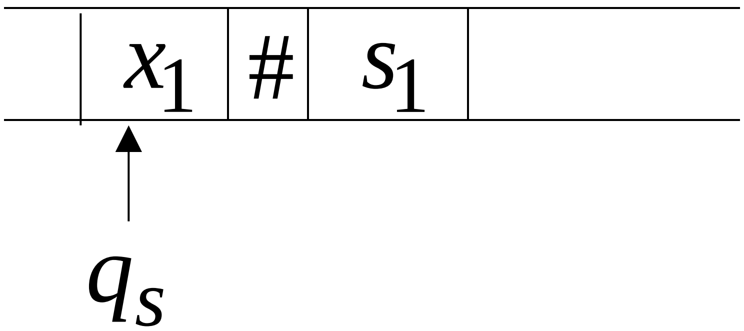
Enumeration Machine

Configuration

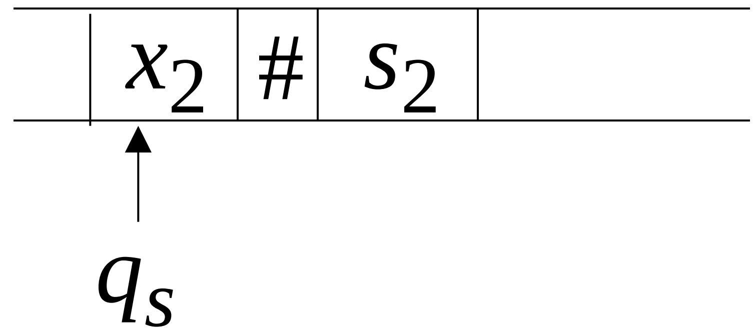
Time 0



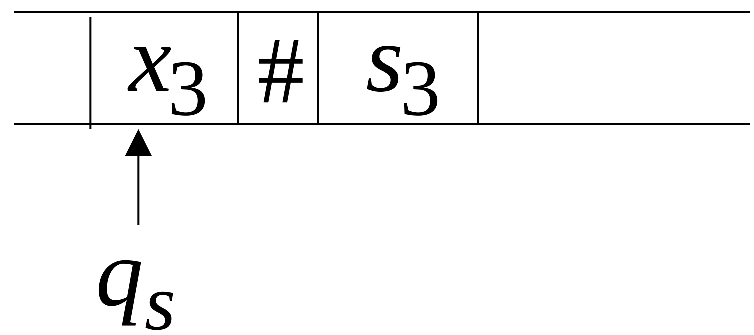
Time t_1



Time t_2



Time t_3



Observation:

If for a set there is an enumeration procedure, then the set is countable

Example:

The set of all strings $\{a,b,c\}^+$ is countable

Proof:

We will describe an enumeration procedure

Naive procedure:

Produce the strings in lexicographic order:

$$a$$

$$aa$$

$$aaa$$

$$aaaa$$

$$\ldots\ldots$$

Doesn't work:

strings starting with b will never be listed

(violates the generation in finite time rule)

Better procedure: **Proper 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:**

$$\left.\begin{array}{l} a \\ b \\ c \end{array}\right\} \text{length 1}$$

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

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

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

Enumeration Procedure:

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

Definition: A set is uncountable if it is not countable

Theorem:

Let S be an infinite countable set

The powerset 2^S of S is uncountable

Proof:

Since S is countable, we can write

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

Elements of S

Elements of the powerset have the form:

$$\{s_1, s_3\}$$

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

……

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

Powerset element	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	$\cdots$

Let's assume (for contradiction) that the powerset is countable.

Then: we can enumerate the elements of the powerset

Powerset element	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	$\cdots$
$\cdots$						

Take the powerset element whose bits are the complements in the diagonal

Cantor's Diagonal Argument

$$\begin{array}{lccccccc} t_1 & & 1 & 0 & 0 & 0 & 0 & \cdots \\ t_2 & & 1 & 1 & 0 & 0 & 0 & \cdots \\ t_3 & & 1 & 1 & 0 & 1 & 0 & \cdots \\ t_4 & & 1 & 1 & 0 & 0 & 1 & \cdots \end{array}$$

New element: $t = 0011\ldots$

(binary complement of diagonal)

The new element t must be some t_i in the powerset

However, that's impossible:

By construction, t differs from each t_i , since their nth digits differ

Hence, t cannot occur in the enumeration.

There is something we can't count.

Contradiction!!!

Since we have a contradiction:

The powerset 2^S of S is uncountable

An Application: Languages

Example Alphabet : $\{a, b\}$

The set of all Strings:

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

infinite and countable

Example Alphabet : $\{a,b\}$

The set of all Strings:

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

infinite and countable

A language is a subset of S :

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

Example Alphabet : $\{a, b\}$

The set of all Strings:

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

infinite and countable

The powerset of S contains all languages:

$$2^S = \{\underbrace{\{\lambda\}}_{L_1}, \underbrace{\{a\}}_{L_2}, \underbrace{\{a, b\}}_{L_3} \underbrace{\{aa, ab, aab\}}_{L_4}, \underbrace{\ldots}_{\cdots}\}$$

uncountable

Languages: uncountable

$$L_1 \quad L_2 \quad L_3 \quad \cdots \quad L_k \quad \cdots$$

$$\downarrow \quad \downarrow \quad \downarrow \quad \quad \downarrow$$

$$M_1 \quad M_2 \quad M_3 \quad \quad ?$$

Turing machines: countable

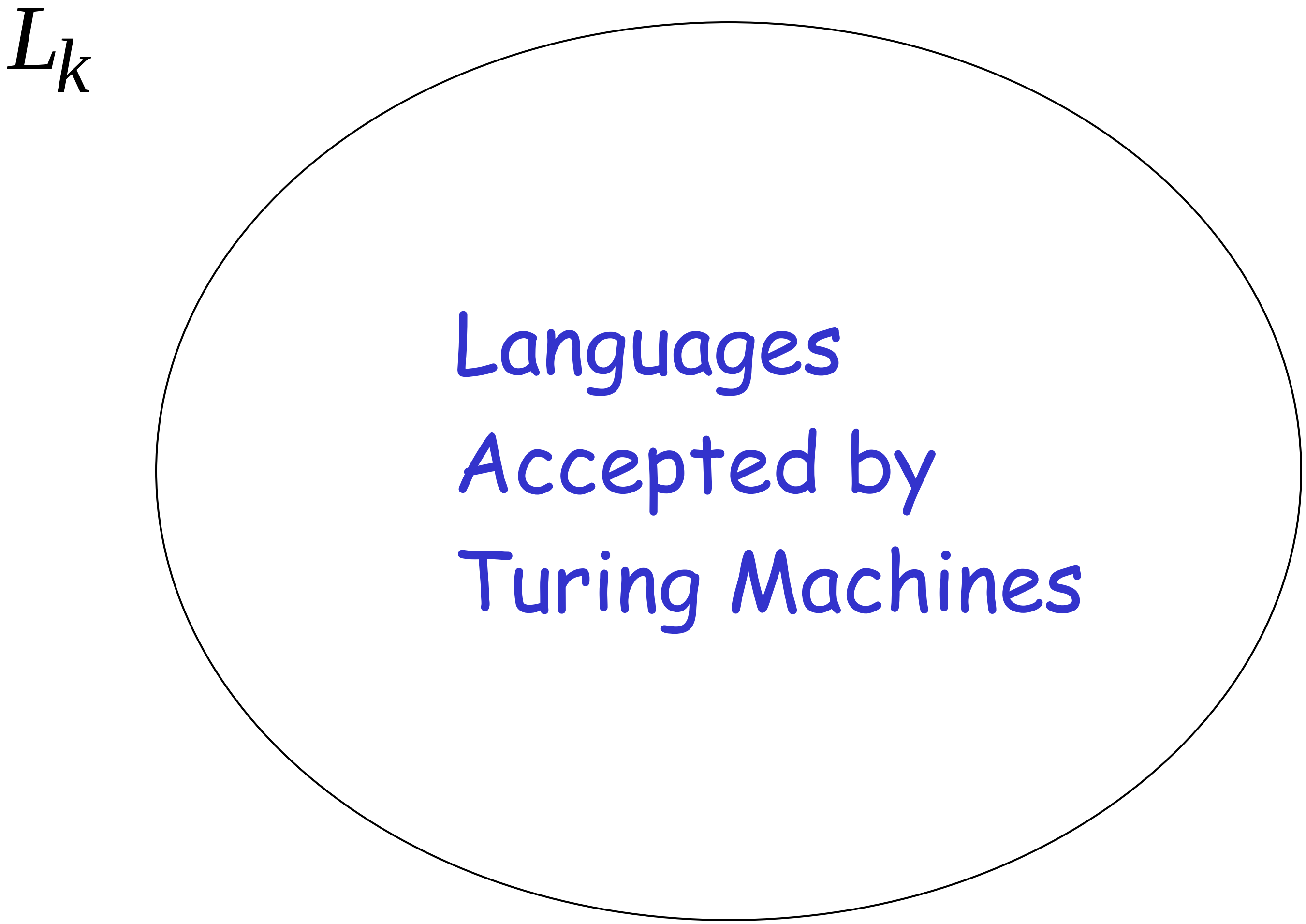
There are more languages
than Turing Machines

Conclusion:

There are some languages not accepted by Turing Machines

(These languages cannot be described by algorithms)

Languages not accepted by Turing Machines



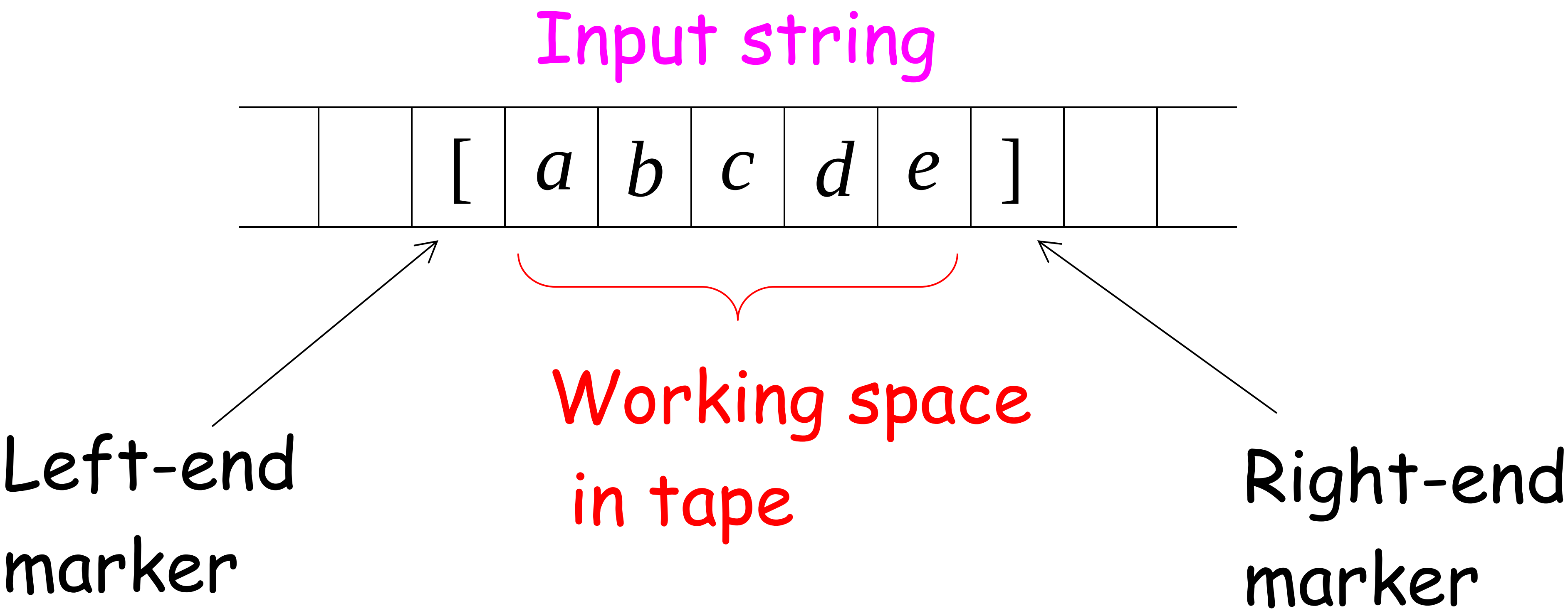
Outline

- Review of last week
- Universal Turing Machine
- Countable/uncountable Sets
- Linear Bounded Automata
- Chomsky Hierarchy and Recursively Enumerable Languages

Linear Bounded Automata (LBAs)
are the same as Turing Machines
with one difference:

The input string tape space
is the only tape space allowed to use

Linear Bounded Automaton (LBA)



All computation is done between end markers

We define LBA's as NonDeterministic

Open Problem:

NonDeterministic LBA's
have same power with
Deterministic LBA's ?

Example languages accepted by LBAs:

$$L = \{a^n b^n c^n\}$$

$$L = \{a^{n!}\}$$

LBA's have more power than NPDA's

LBA's have also less power than Turing Machines

Context-Sensitive Grammars:

Productions

$$u \rightarrow v$$

String of variables and terminals

String of variables and terminals

and: $|u| \leq |v|$

The language $\{a^n b^n c^n\}$

is context-sensitive:

$$S \rightarrow abc \mid aAbc$$

$$Ab \rightarrow bA$$

$$Ac \rightarrow Bbcc$$

$$bB \rightarrow Bb$$

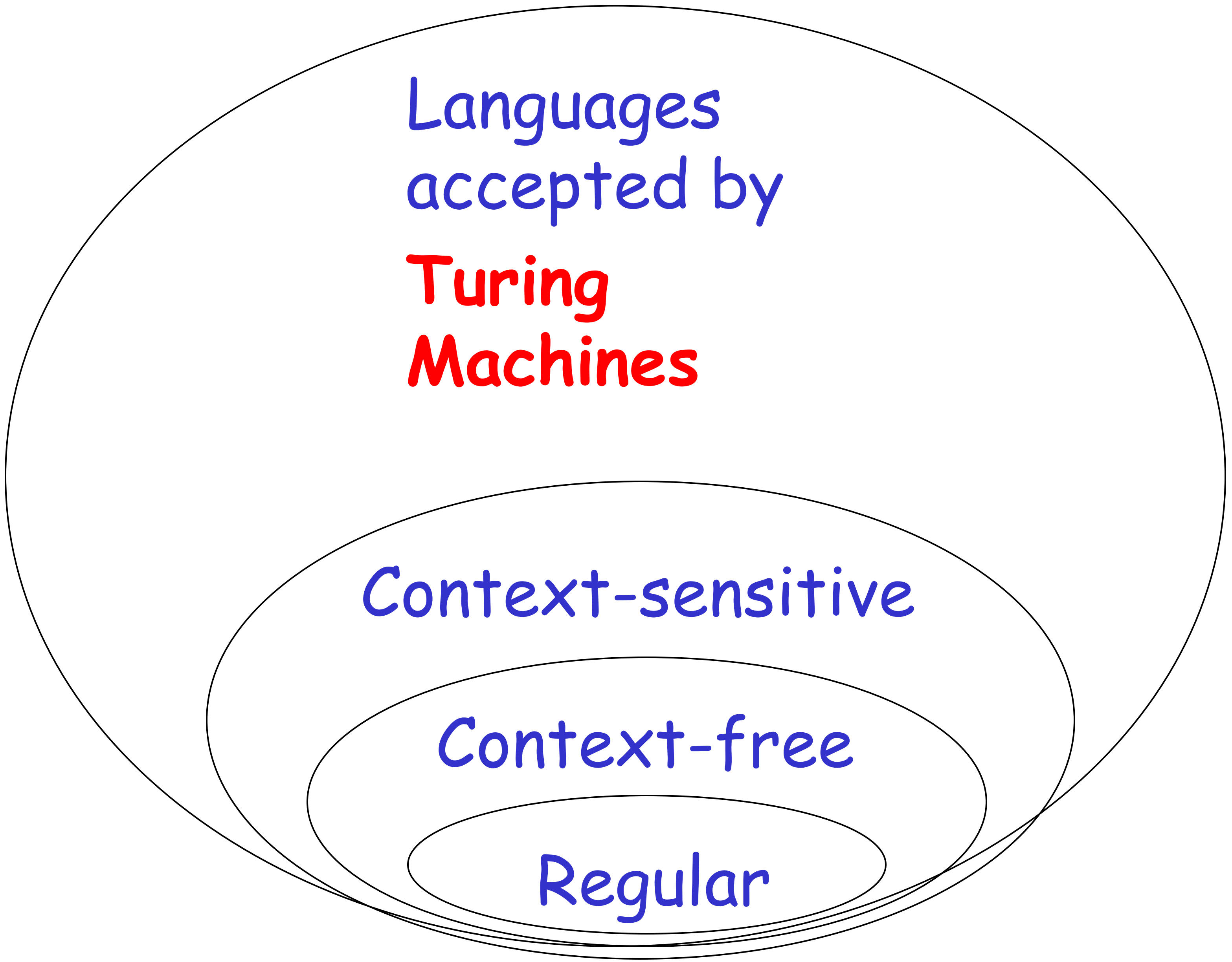
$$aB \rightarrow aa \mid aaA$$

Theorem:

A language L is context sensistive

if and only if

L is accepted by a Linear-Bounded automaton

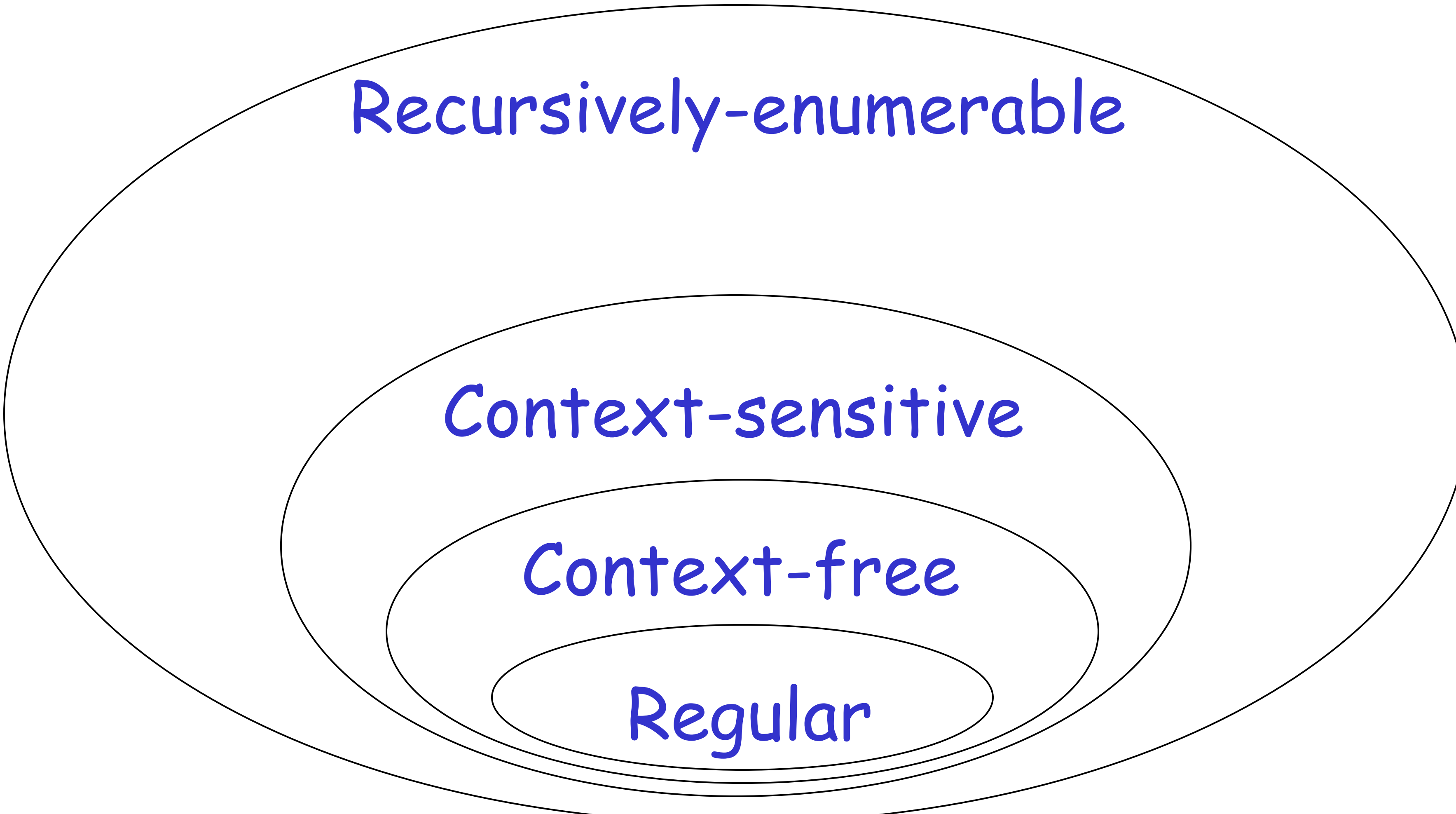

Languages
accepted by
Turing
Machines
Context-sensitive
Context-free
Regular

Outline

- Review of last week
- Universal Turing Machine
- Countable/uncountable Sets
- Linear Bounded Automata
- Chomsky Hierarchy and Recursively Enumerable Languages

The Chomsky Hierarchy

Non-recursively enumerable



Unrestricted Grammars:

Productions

$$u \rightarrow v$$

String of variables and terminals

String of variables and terminals

Example unrestricted grammar:

$$S \rightarrow aBc$$

$$aB \rightarrow cA$$

$$Ac \rightarrow d$$

Recursively Enumerable 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

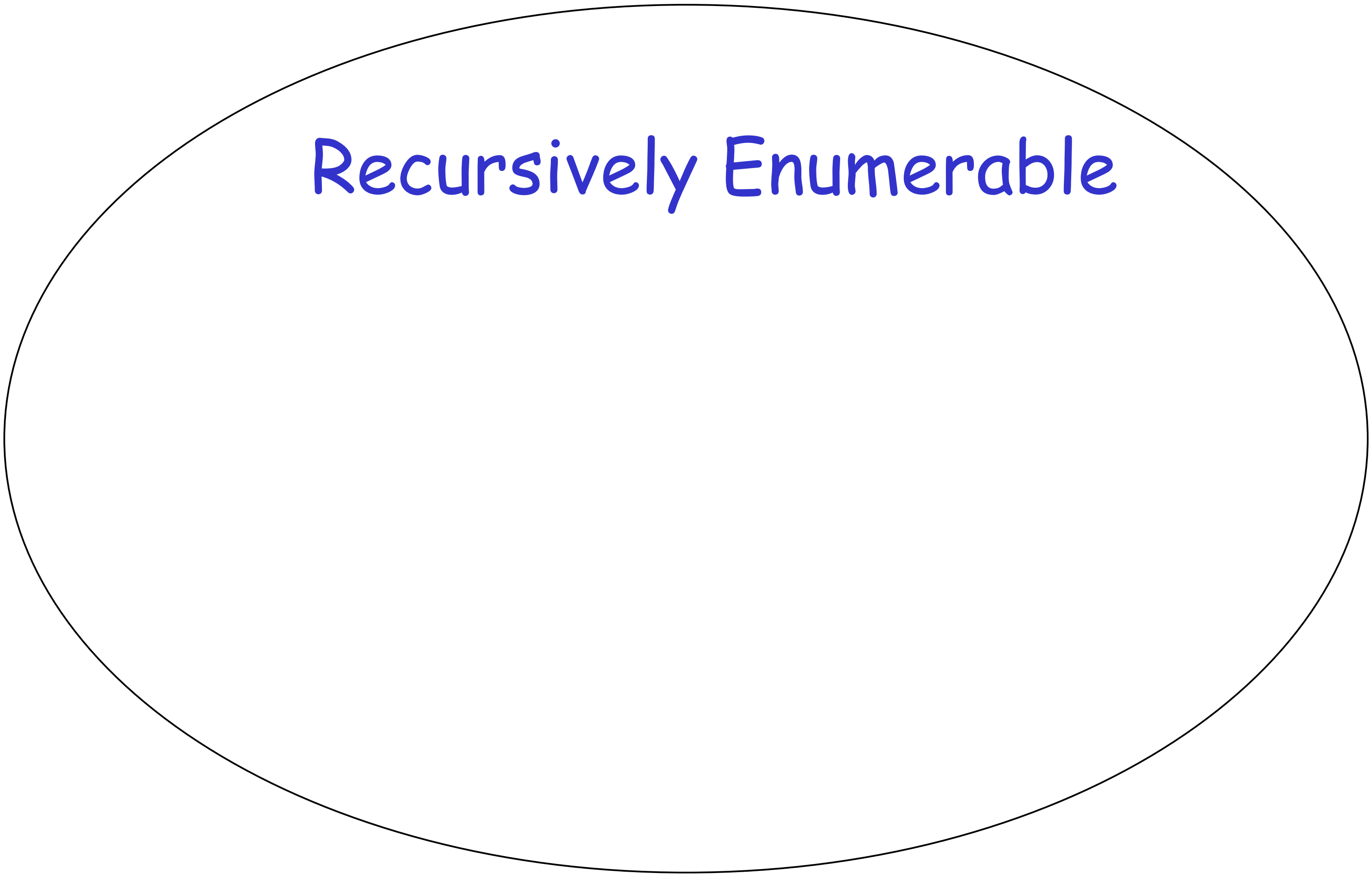
or loops forever

We will prove:

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

A Language which
is not
Recursively Enumerable

Non 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	$\cdots$

Answer: $M_k \neq M_1$

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

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

	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:

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

or

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

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

Turing acceptable languages
and
Enumeration Procedures

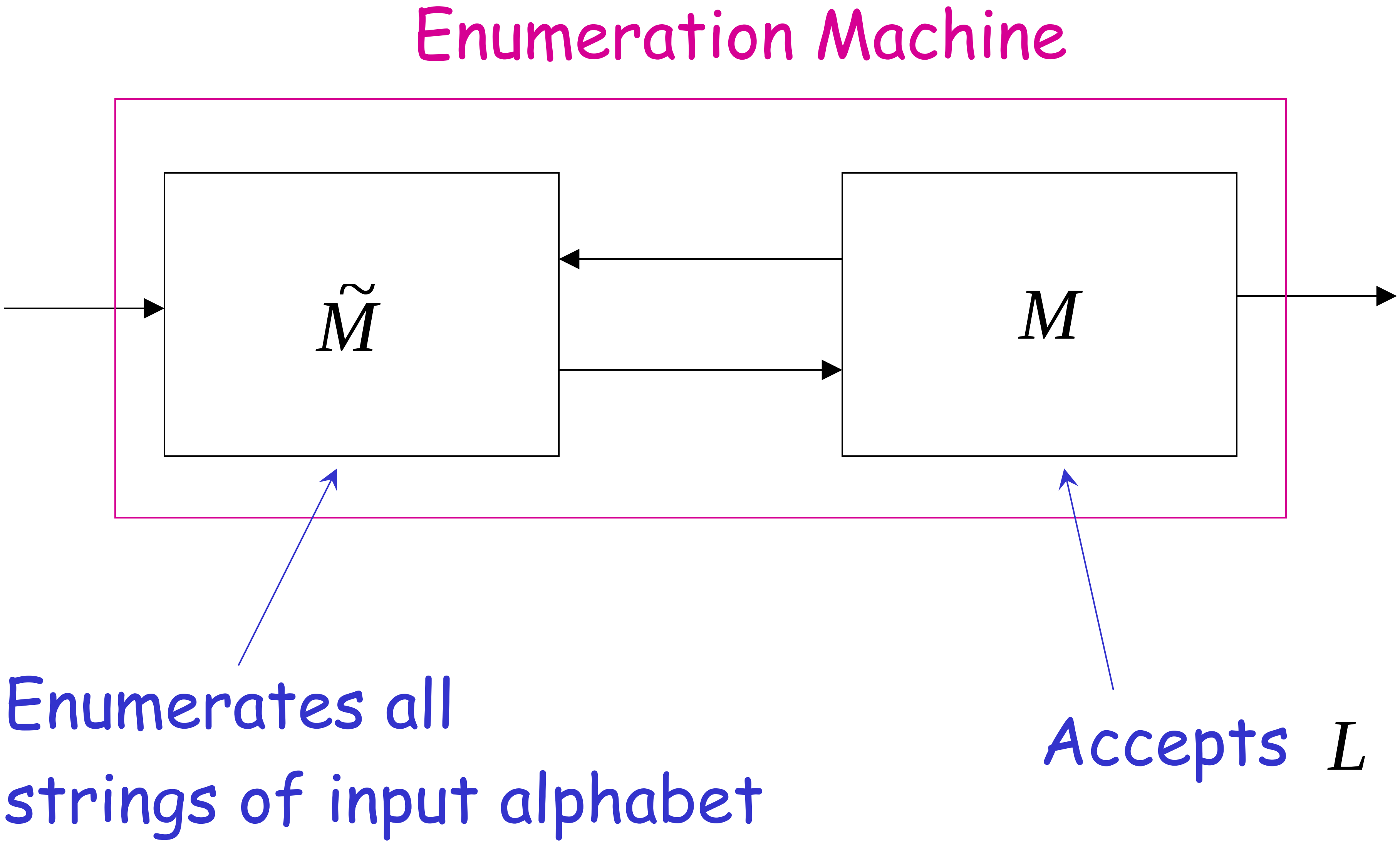
We will prove:

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

Theorem:

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

Proof:


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

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

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

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

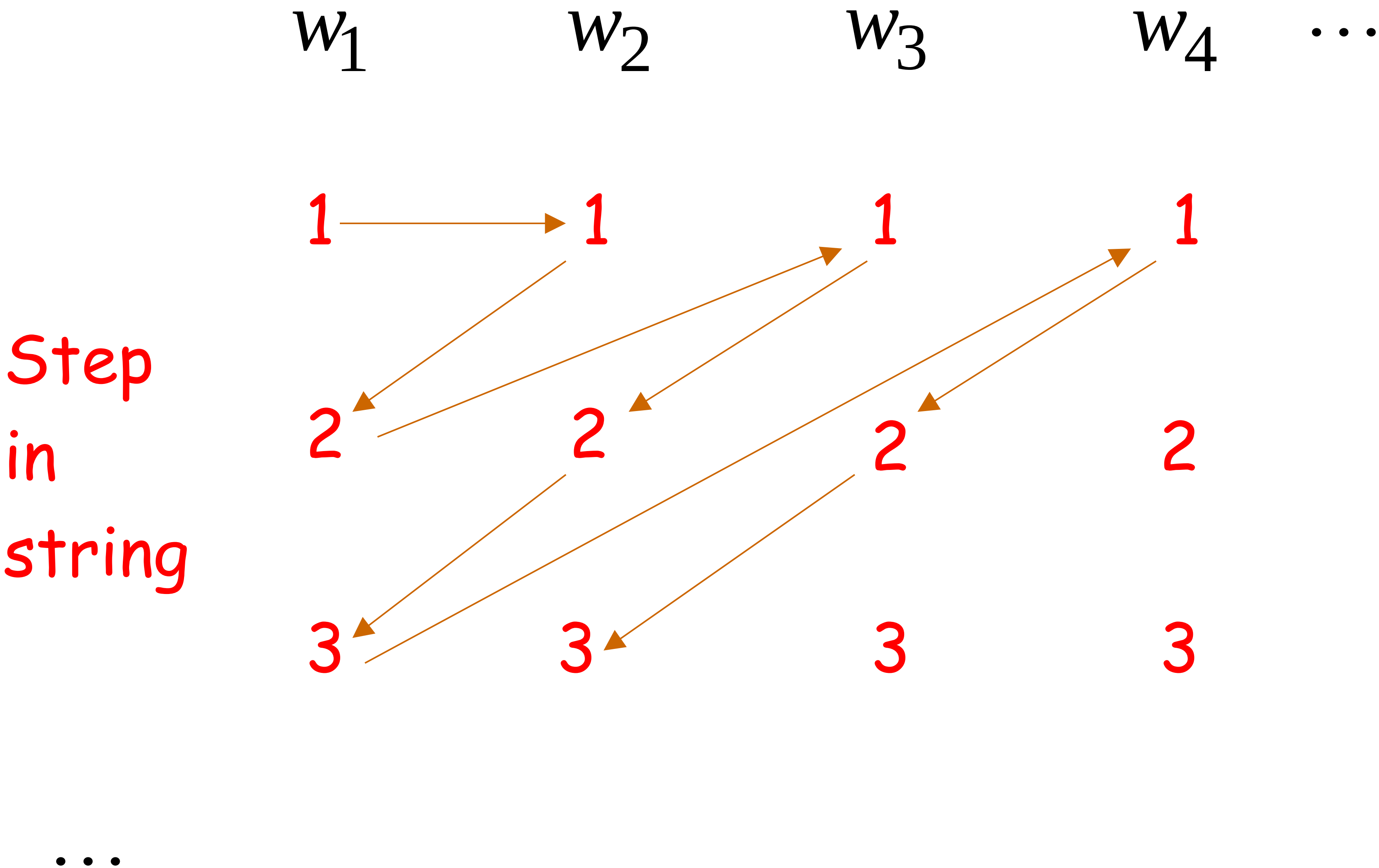
$\tilde{M}$ Generates third string w_3

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
2 2 2 2
3 3 3 3
Step in string
…

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

End of Proof

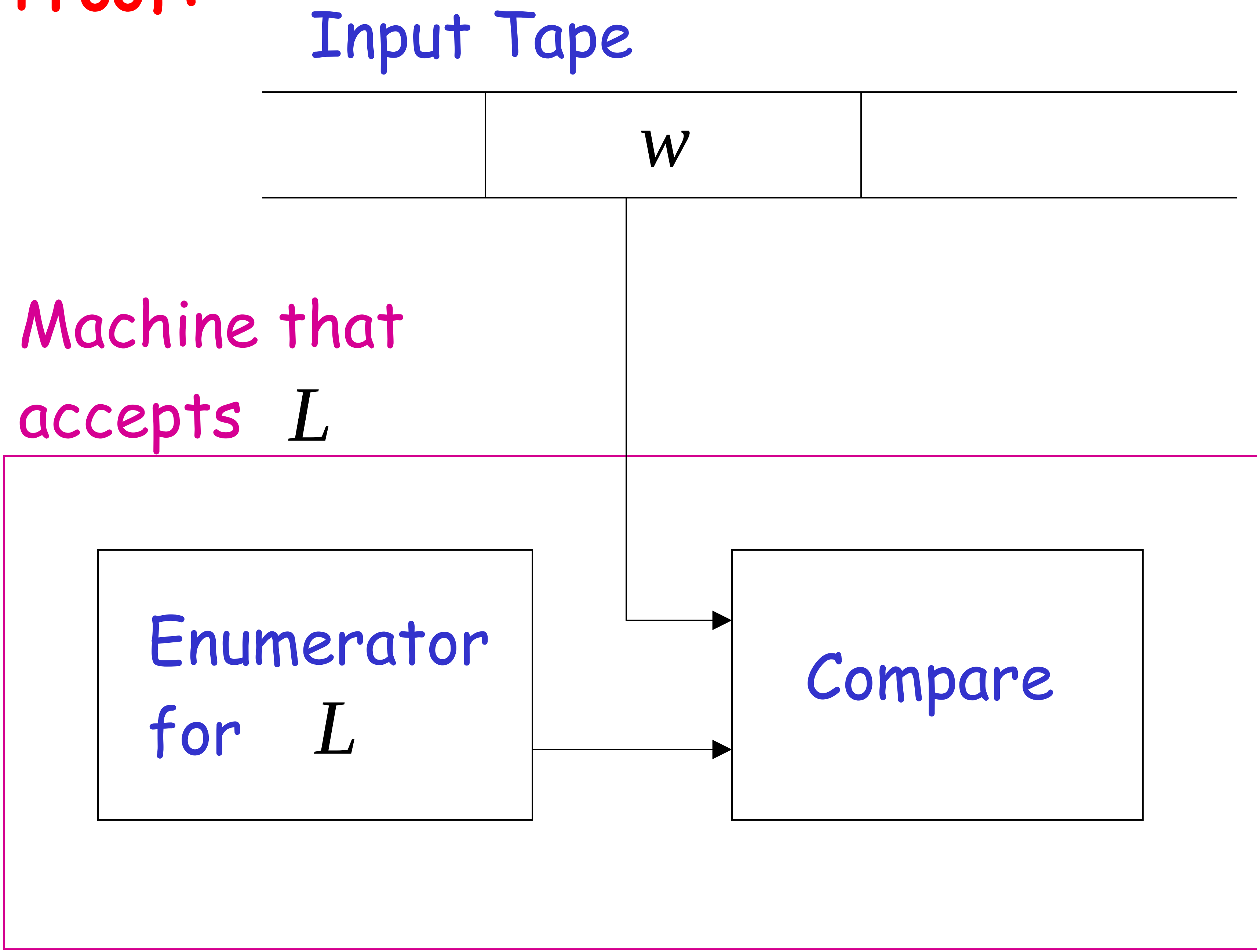
Theorem:

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

Theorem:

A language L is recursively enumerable if and only if L is generated by an unrestricted grammar

The Chomsky Hierarchy