Recursively Enumerable and Recursive Languages

Turing Machine

There are *three* possible outcomes of executing a Turing machine over a given input.

The Turing machine may

- Halt and accept the input
- Halt and reject the input
- Never halt. (loops for ever)

Definition:

A language is **recursively enumerable** if there is a Turing machine that accepts it

Also known as: *Turing Recognizable* languages or *Turing Acceptable* languages

Let L be a recursively enumerable language

and M the Turing Machine that accepts it

For any string w :

if $w \in L$ then M halts in an accept state

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

or loops forever

Definition:

A language is **recursive**
if there is a Turing machine accepts it
and the machine halts on every input string

Also known as *decidable languages*

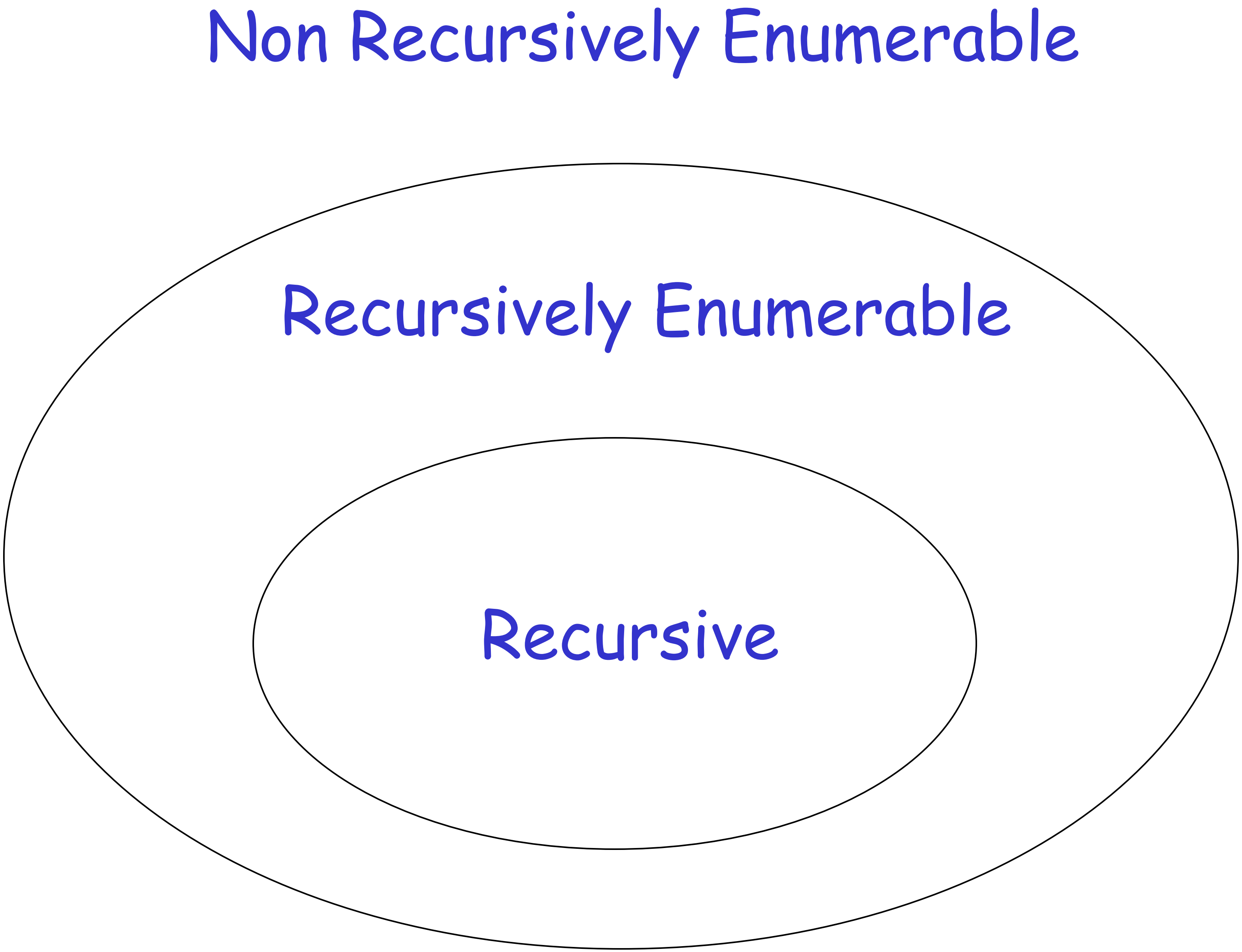
Let L be a recursive language

Then there is a Turing machine M that accepts L such that:

For any string w :

if $w \in L$ then M halts in an accept state

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

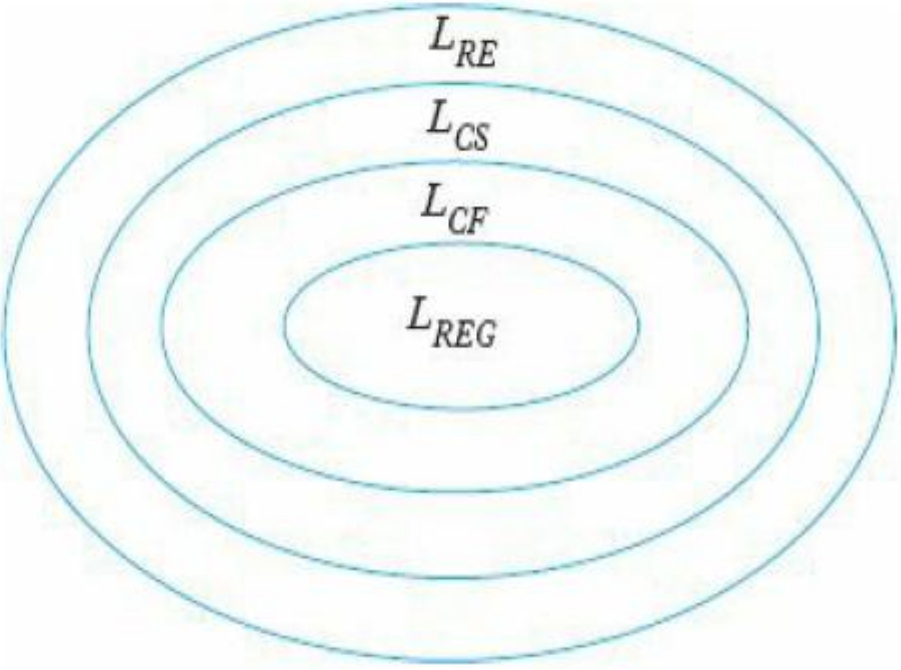

Non Recursively Enumerable
Recursively Enumerable
Recursive

Decidable & Undecidable

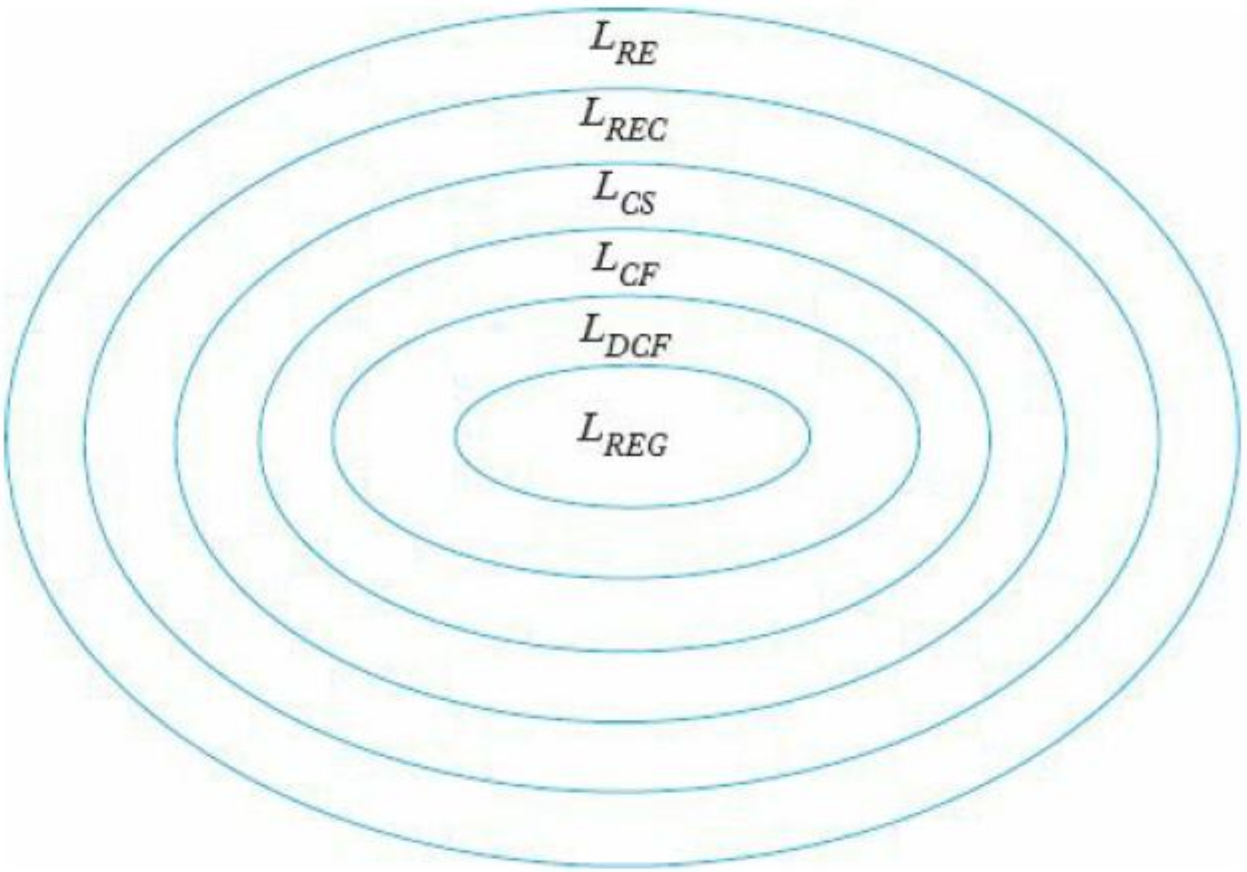
Language L is Decidable if it is a recursive language.

It is Undecidable if it not a recurisve language

Chmosky Hierarchy



Chmosky Hierarchy



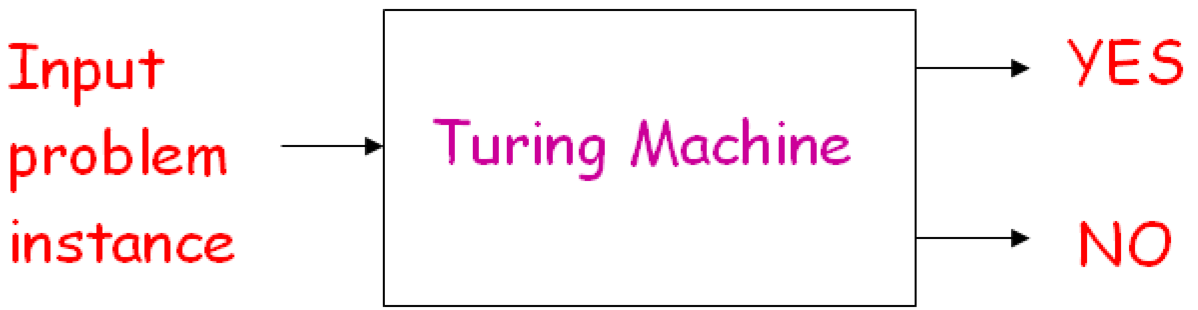
Decidability

Consider problems with answer YES or NO

Examples:

- Does Machine M have three states ?
- Is string w a binary number?
- Does DFA M accept any input?

The Turing machine that decides (solves)
a problem answers YES or NO
for each instance of the problem



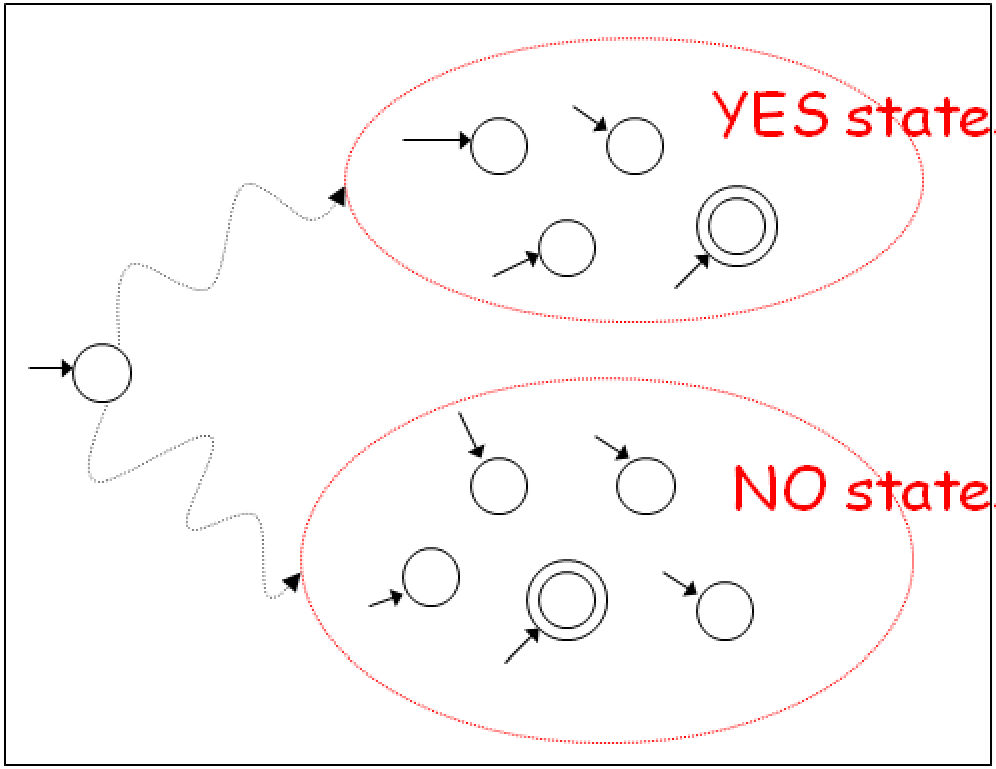
The machine that decides (solves) a problem:

- If the answer is YES
  then halts in a yes state

- If the answer is NO
  then halts in a no state

These states may not be final states

Turing Machine that decides a problem



YES and NO states are halting states

Difference between

Recursive Languages and Decidable problems

For decidable problems:

The YES states may not be final states

Some problems are undecidable:

which means:
there is no Turing Machine that
solves all instances of the problem

A simple undecidable problem:

The membership problem

The Membership Problem

Input:
- Turing Machine M
- String w

Question: Does M accept w ?

$$w \in L(M)?$$

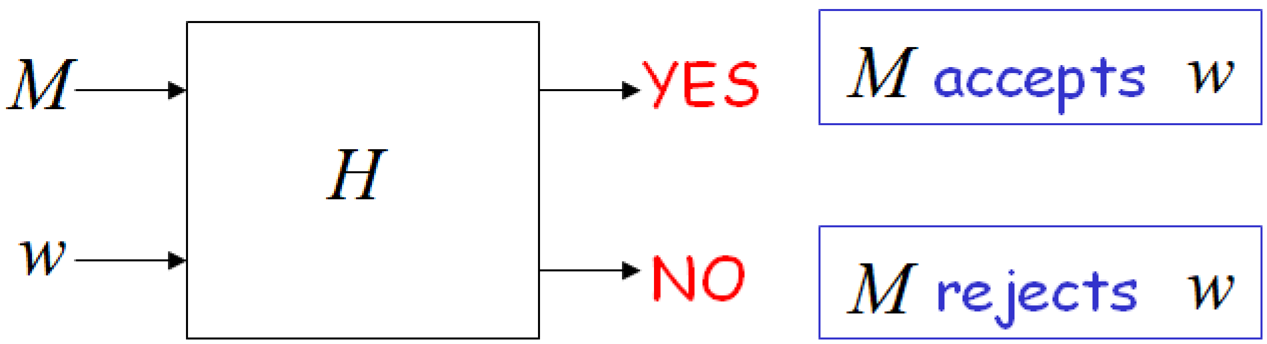
Theorem:

The membership problem is undecidable

(there are M and w for which we cannot decide whether $w \in L(M)$)

Proof: Assume for contradiction that the membership problem is decidable

Thus, there exists a Turing Machine H
that solves the membership problem



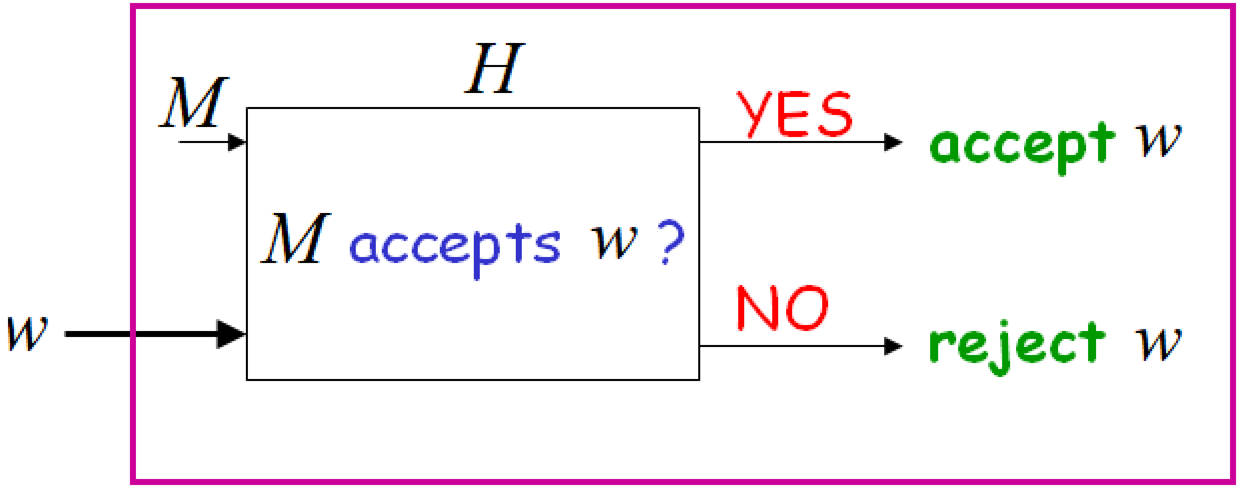
Let L be a recursively enumerable language

Let M be the Turing Machine that accepts L

We will prove that L is also recursive:

we will describe a Turing machine that accepts L and halts on any input

Turing Machine that accepts L
and halts on any input



Therefore, L is recursive

Since L is chosen arbitrarily, every recursively enumerable language is also recursive

But there are recursively enumerable languages which are not recursive

Contradiction!!!!

Therefore, the membership problem
is undecidable

Another undecidable problem:

The halting problem

The Halting Problem

Input:
- Turing Machine M
- String w

Question: Does M halt on input w?

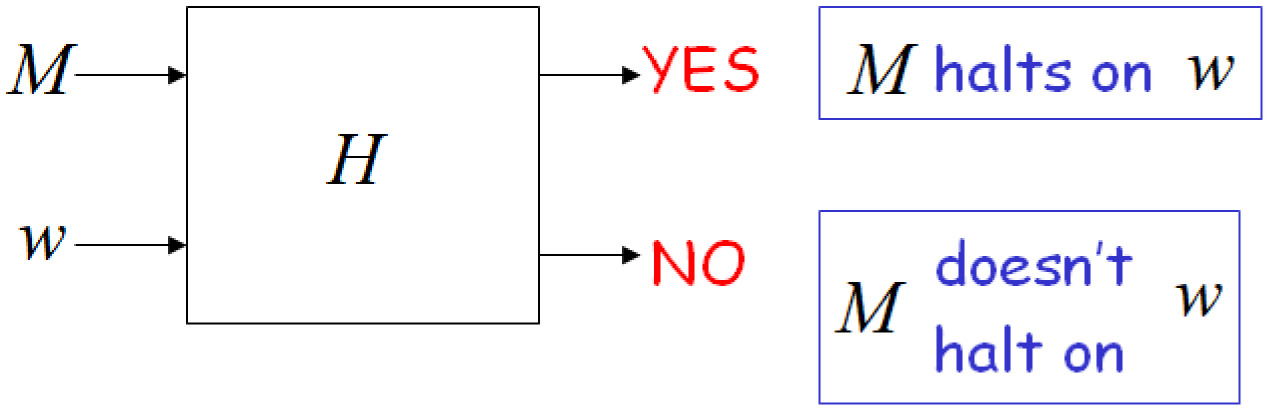
Theorem:

The halting problem is undecidable

(there are M and w for which we cannot decide whether M halts on input w)

Proof: Assume for contradiction that the halting problem is decidable

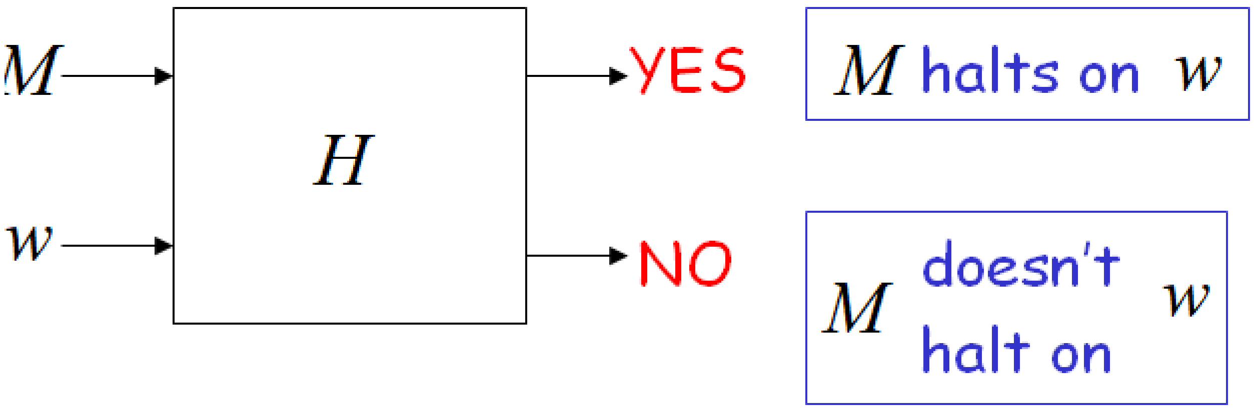
Thus, there exists Turing Machine H
that solves the halting problem



Proof

If the halting problem was decidable then every recursively enumerable language would be recursive

There exists Turing Machine H
that solves the halting problem



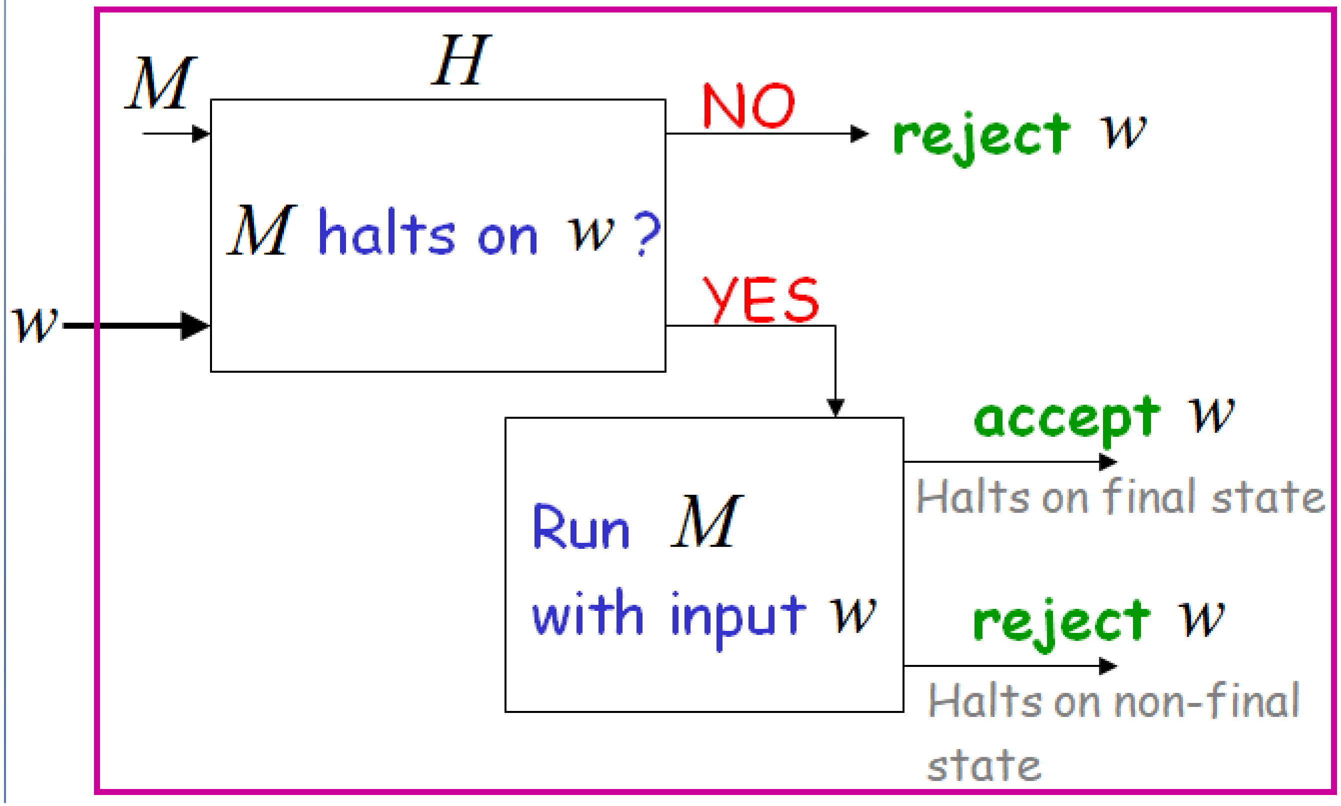
Let L be a recursively enumerable language

Let M be the Turing Machine that accepts L

We will prove that L is also recursive:

we will describe a Turing machine that accepts L and halts on any input

Turing Machine that accepts L
and halts on any input



Therefore L is recursive

Since L is chosen arbitrarily, every recursively enumerable language is also recursive

But there are recursively enumerable languages which are not recursive

Contradiction!!!!

Therefore, the halting problem is undecidable

- Recursively enumerable languages are also known as type 0 languages.
- Context-sensitive languages are also known as type 1 languages.
- Context-free languages are also known as type 2 languages.
- Regular languages are also known as type 3 languages.

- Definition: Let L be a language. Then L is recursively enumerable if there exists a TM M such that L = L(M).

  - If L is r.e. then L = L(M) for some TM M, and
    - If x is in L then M halts in a final (accepting) state.
    - If x is not in L then M may halt in a non-final (non-accepting) state or no transition is available, or loop forever.

- Definition: Let L be a language. Then L is recursive if there exists a TM M such that L = L(M) and M halts on all inputs.

  - If L is recursive then L = L(M) for some TM M, and
    - If x is in L then M halts in a final (accepting) state.
    - If x is not in L then M halts in a non-final (non-accepting) state or no transition is available (does not go to infinite loop).

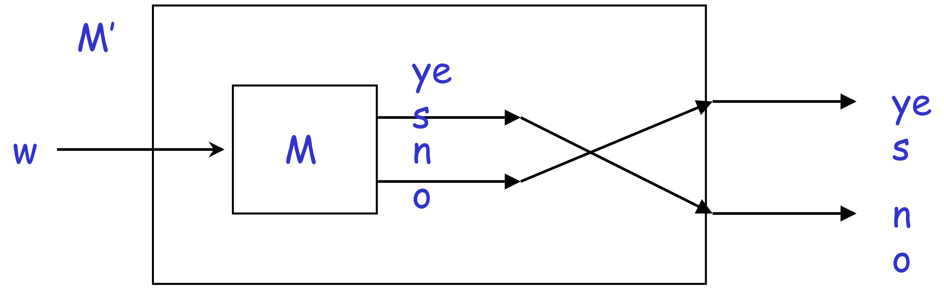
Notes:

  - The set of all recursive languages is a subset of the set of all recursively enumerable languages

Closure Properties
for
Recursive and Recursively Enumerable Languages

Theorem 1: The recursive languages are closed with respect to complementation, i.e., if L is a recursive language, then so is $\overline{L} = \Sigma^* - L$

Proof: Let M be a TM such that L = L(M) and M always halts. Construct TM M' as follows:



Note That:

M' accepts iff M does not

M' always halts since M always halts

From this it follows that the complement of L is recursive. •

Question: How is the construction achieved? Do we simply complement the final states in the TM? No! A string in L could end up in the complement of L.

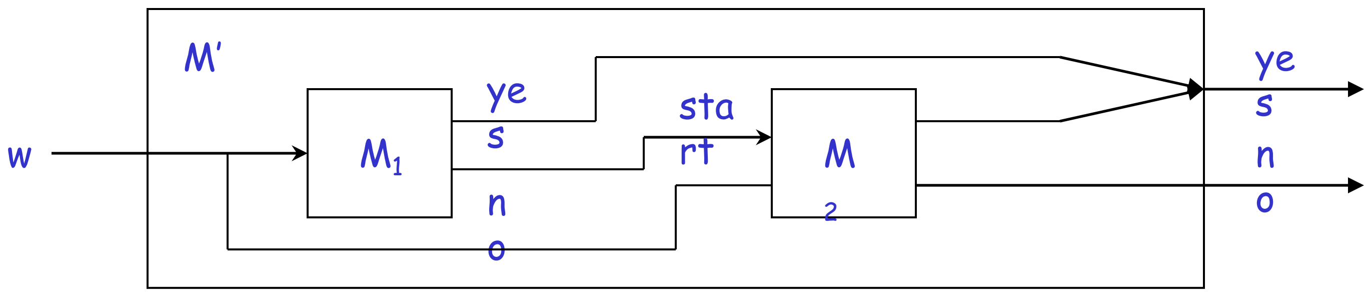
Suppose q5 is an accepting state in M, but q0 is not.

If we simply complemented the final and non-final states, then q0 would be an accepting state in M' but q5 would not.

Since q0 is an accepting state, by definition all strings are accepted by M'

Theorem 2: The recursive languages are closed with respect to union, i.e., if L1 and L2 are recursive languages, then so is $L_3 = L_1 \cup L_2$

Proof: Let M1 and M2 be TMs such that L1 = L(M1) and L2 = L(M2) and M1 and M2 always halts. Construct TM M' as follows:



Note That:

L(M') = L(M1) U L(M2)

L(M') is a subset of L(M1) U L(M2)

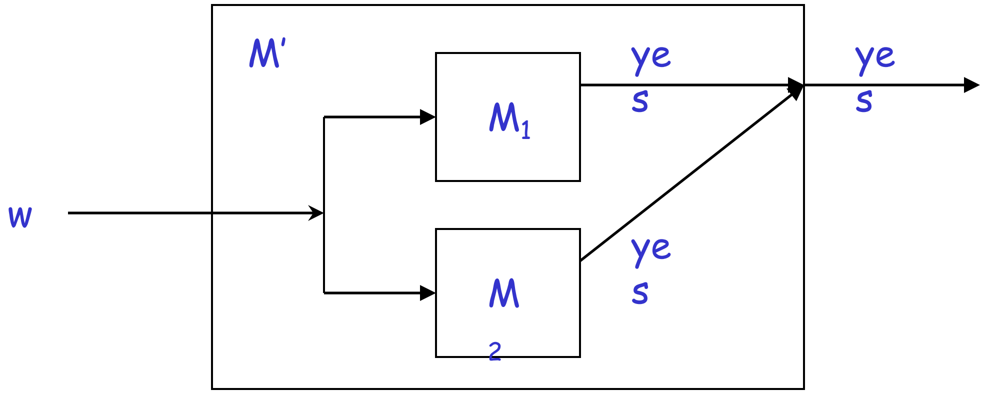
L(M1) U L(M2) is a subset of L(M')

M' always halts since M1 and M2 always halt

It follows from this that $L_3 = L_1 \cup L_2$ is recursive. •

Theorem 3: The recursive enumerable languages are closed with respect to union, i.e., if L1 and L2 are recursively enumerable languages, then so is $L_3 = L_1 \cup L_2$

Proof: Let M1 and M2 be TMs such that L1 = L(M1) and L2 = L(M2). Construct M' as follows:



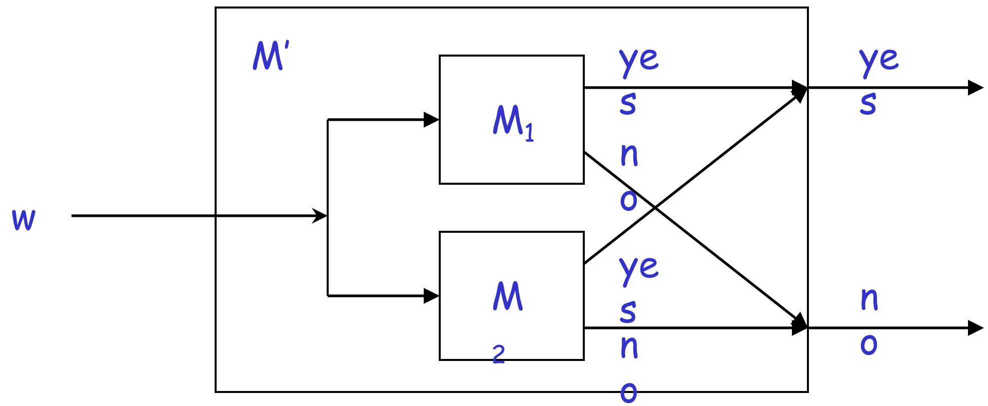
Note That:

- L(M') = L(M1) U L(M2)
  - L(M') is a subset of L(M1) U L(M2)
  - L(M1) U L(M2) is a subset of L(M')
- M' halts and accepts iff M1 or M2 halts and accepts

It follows from this that $L_3 = L_1 \cup L_2$ is recursively enumerable.

Question: How do you run two TMs in parallel?

Suppose, M1 and M2 had outputs for "no" in the previous construction, and these were transferred to the "no" output for M′



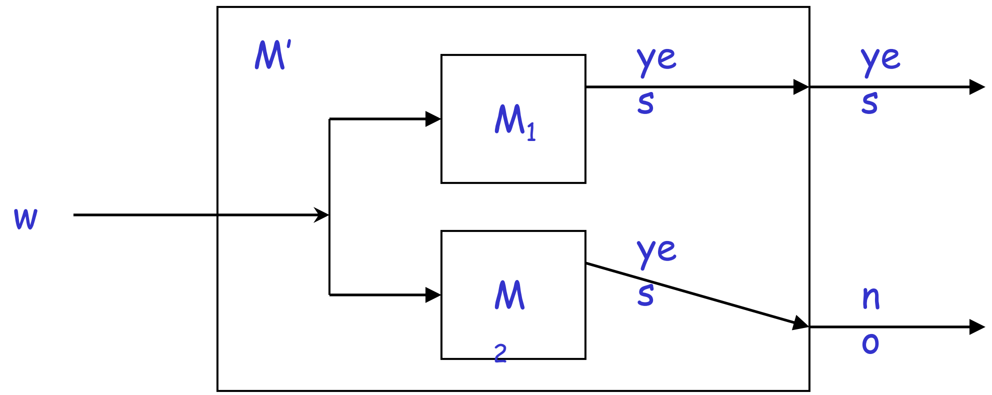
Question: What would happen if w is in L(M1) but not in L(M2)?

Answer: You could get two outputs – one "yes" and one "no."

At least M1 will halt and answer accept, M2 may or may not halt.
As before, for the sake of convenience the "no" output will be ignored.

- Theorem 4: If L and $\overline{L}$ are both recursively enumerable then L (and therefore $\overline{L}$) is recursive.

Proof: Let M1 and M2 be TMs such that L = L(M1) and L= L(M2).
Construct M' as follows:



- Note That:
  - L(M') = L
    - L(M') is a subset of L
    - L is a subset of L(M')
  - M' is TM for L
  - M' always halts since either M1 or M2 halts for any given string
  - M' shows that L is recursive

It follows from this that L (and therefore its' complement) is recursive.

So, $\overline{L}$ is also recursive (we proved it before). •

The Post Correspondence Problem

Some undecidable problems for context-free languages:

- Is $L(G_1) \cap L(G_2) = \varnothing$?

  G_1, G_2 are context-free grammars

- Is context-free grammar G ambiguous?

We need a tool to prove that the previous problems for context-free languages are <u>undecidable</u>:

The Post Correspondence Problem

There is a **Post Correspondence Solution** if there is a sequence $i, j, \ldots, k$ such that:

PC-solution: $w_i w_j \cdots w_k = v_i v_j \cdots v_k$

Indices may be repeated or omitted

Some undecidable problems for context-free languages:

- Is $L(G_1) \cap L(G_2) = \varnothing$?

  G_1, G_2 are context-free grammars

- Is context-free grammar G ambiguous?

The Post Correspondence Problem

Input: Two sets of n strings

$$A = w_1, w_2, \ldots, w_n$$

$$B = v_1, v_2, \ldots, v_n$$

There is a Post Correspondence Solution
if there is a sequence $i, j, \ldots, k$ such that:

PC-solution: $$w_i w_j \cdots w_k = v_i v_j \cdots v_k$$

Indices may be repeated or omitted

Example:

	w_1	w_2	w_3
$A:$	100	11	111
	v_1	v_2	v_3
$B:$	001	111	11

PC-solution: 2,1,3

$$w_2 w_1 w_3 = v_2 v_1 v_3$$

11100111

Example

	w	x
	A	B
1	1	111
2	10111	10
3	10	0

PC-solution: 2 1 1 3

$$w_2 w_1 w_1 w_3 = x_2 x_1 x_1 x_3$$

Example

w x

Click to enlarge

	A	B
1	100	1
2	0	100
3	1	00

PC-solution: problem does not have solution.

Example:

	w_1	w_2	w_3
$A:$	100	11	111
	v_1	v_2	v_3
$B:$	001	111	11

PC-solution: 2,1,3

$$w_2 w_1 w_3 = v_2 v_1 v_3$$

11100111

Example:

	w_1	w_2	w_3
$A:$	00	001	1000
	v_1	v_2	v_3
$B:$	0	11	011

There is no solution

The Modified Post Correspondence Problem

Inputs: $A = w_1, w_2, \ldots, w_n$

$$B = v_1, v_2, \ldots, v_n$$

MPC-solution: $1, i, j, \ldots, k$

$$w_1 w_i w_j \cdots w_k = v_1 v_i v_j \cdots v_k$$

Example:

	w_1	w_2	w_3
$A:$	11	111	100

	v_1	v_2	v_3
$B:$	111	11	001

MPC-solution: 1,3,2 $\qquad w_1 w_3 w_2 = v_1 v_3 v_2$

11100111

1. The MPC problem is undecidable

(by reducing the membership to MPC)

2. The PC problem is undecidable

(by reducing MPC to PC)

Theorem: The MPC problem is undecidable

Proof: By reducing the membership problem to the MPC problem

Membership problem

Input: Turing machine M
string w

Question: $w \in L(M)$?

Undecidable

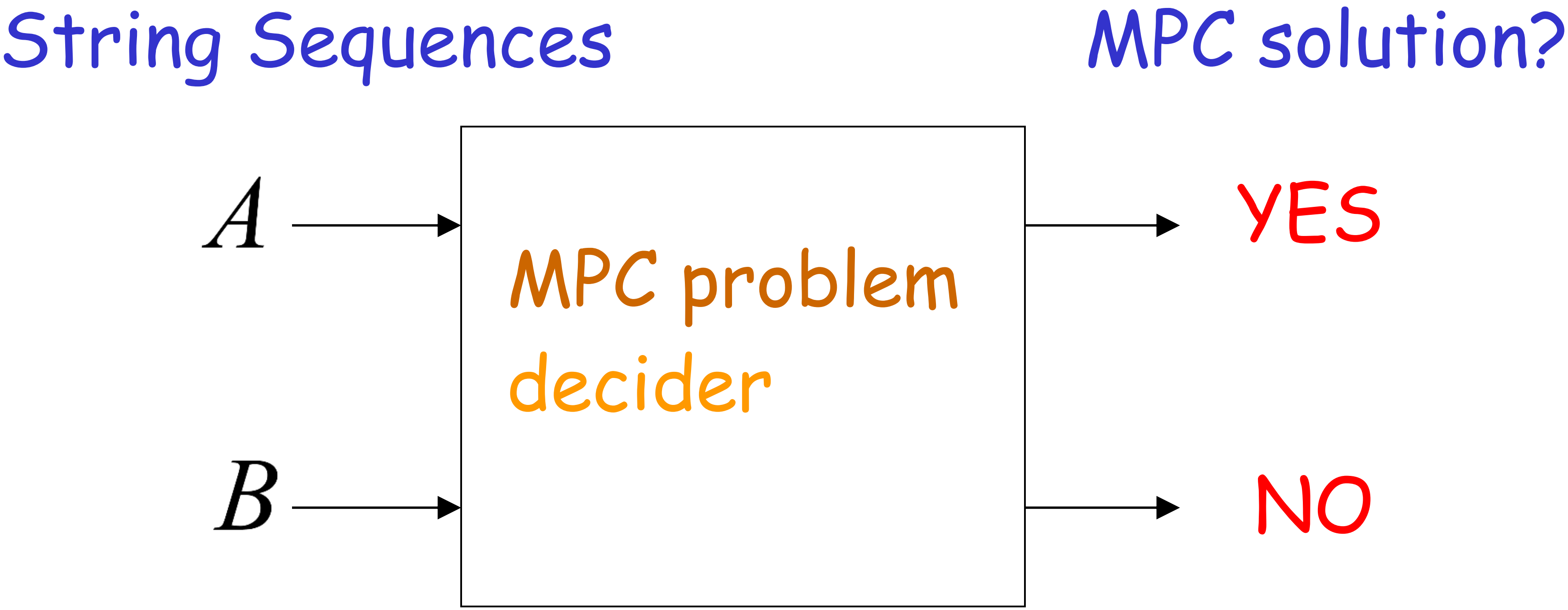
Membership problem

Input: unrestricted grammar G
string w

Question: $w \in L(G)?$

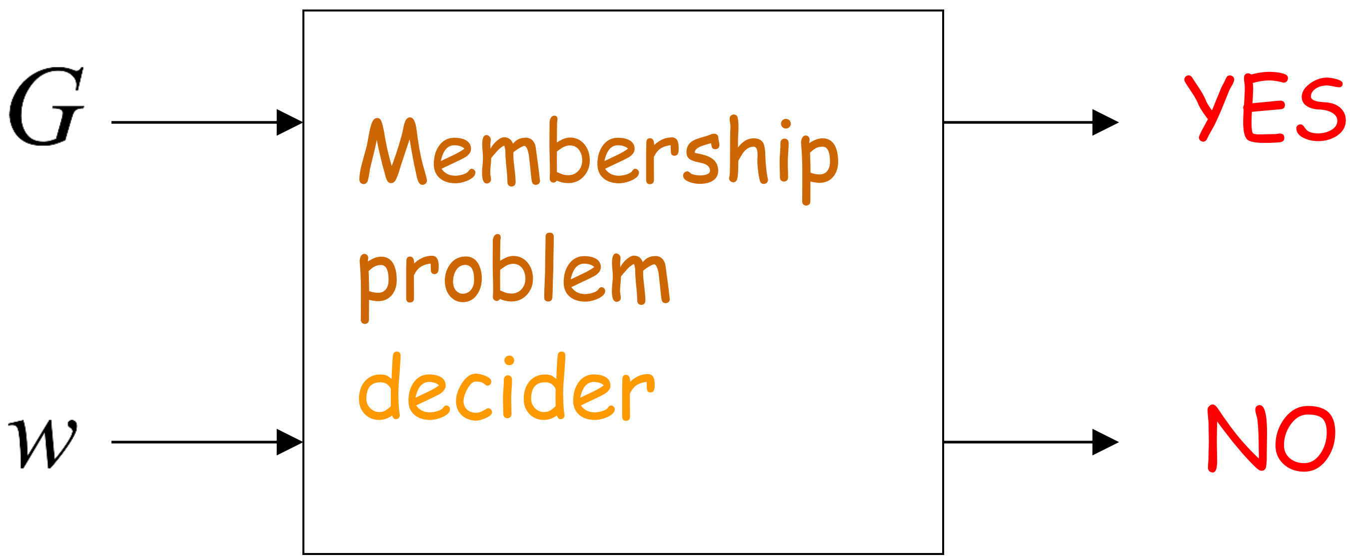
<u>Undecidable</u>

Suppose we have a decider for the MPC problem

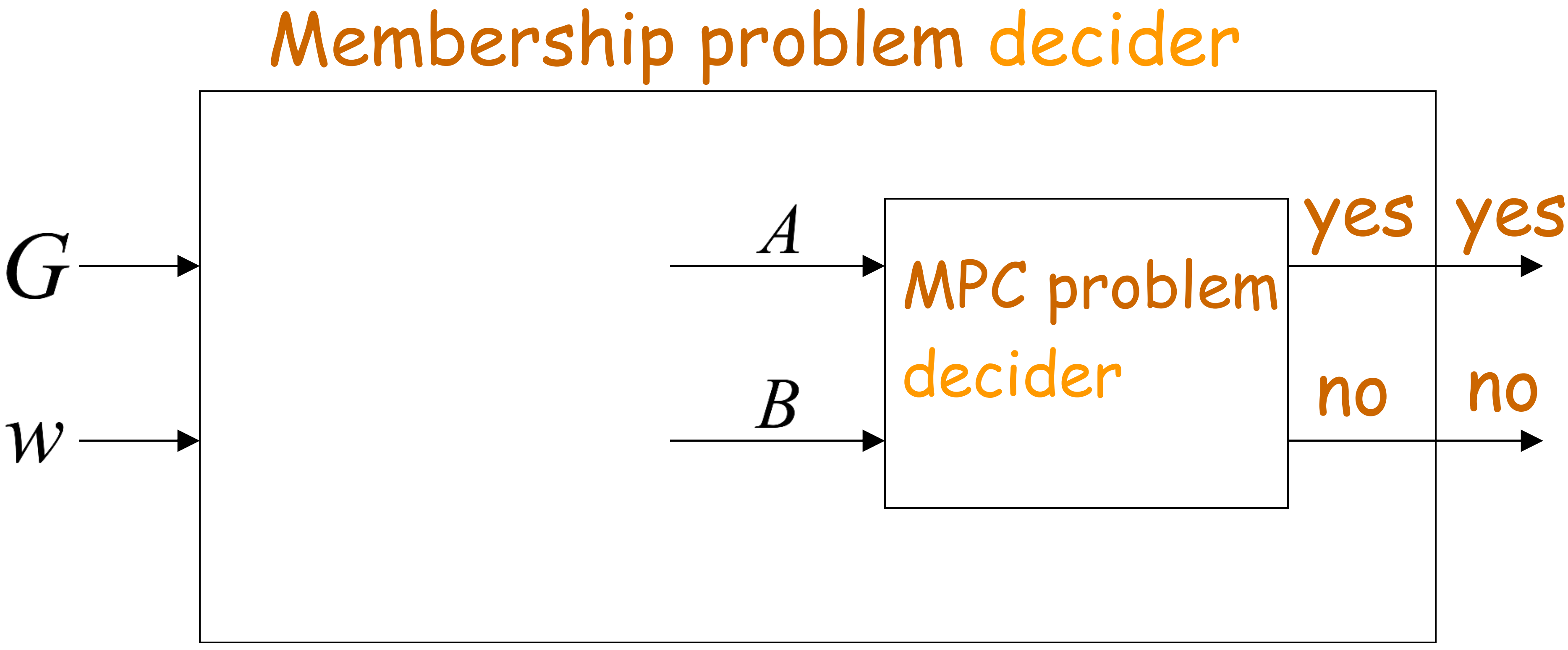


We will build a decider for the membership problem

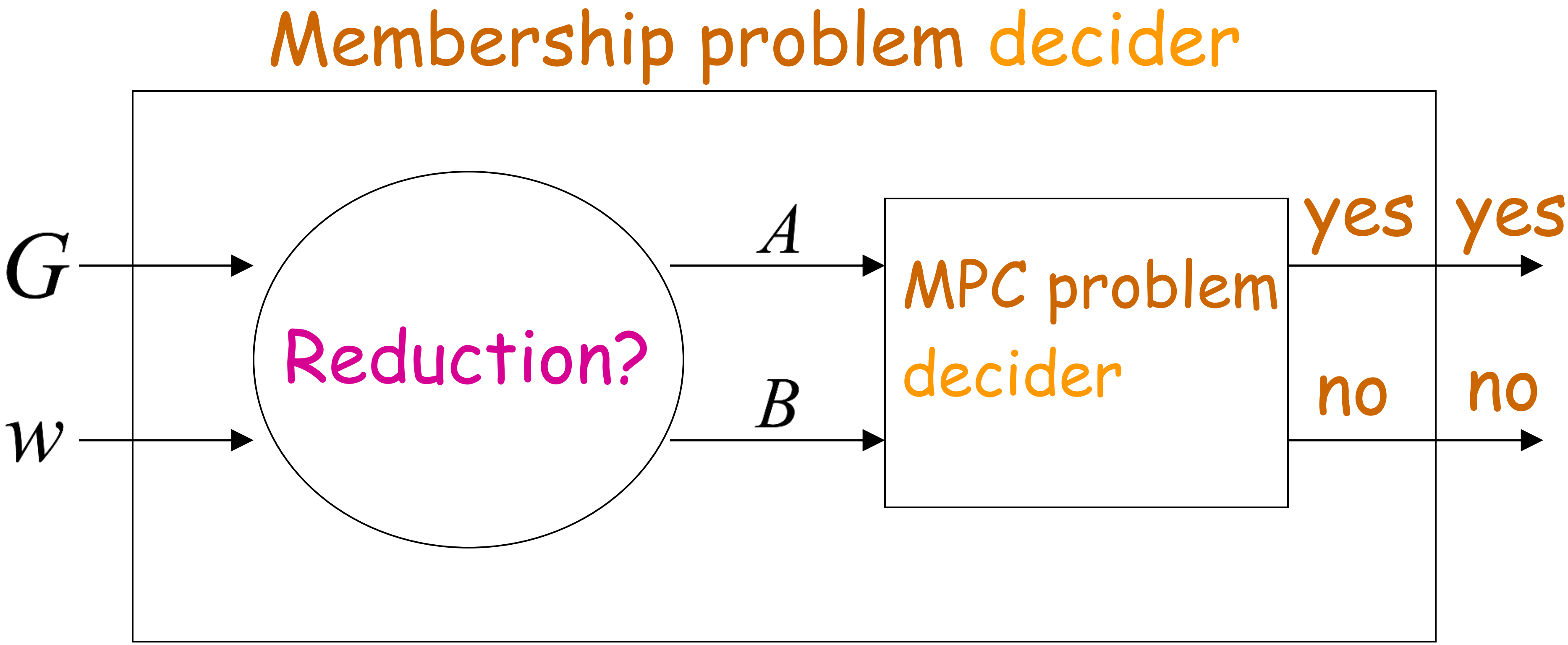
$$w \in L(G)?$$



The reduction of the membership problem to the MPC problem:



We need to convert the input instance of one problem to the other



NDTM: Non deterministic Turing Machine

A *nondeterministic* Turing machine (NTM) differs from the deterministic variety we have been studying by having a transition function δ such that for each state q and tape symbol X, $\delta(q, X)$ is a set of triples

$$\{(q_1, Y_1, D_1),\ (q_2, Y_2, D_2), \ldots, (q_k, Y_k, D_k)\}$$