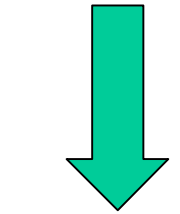
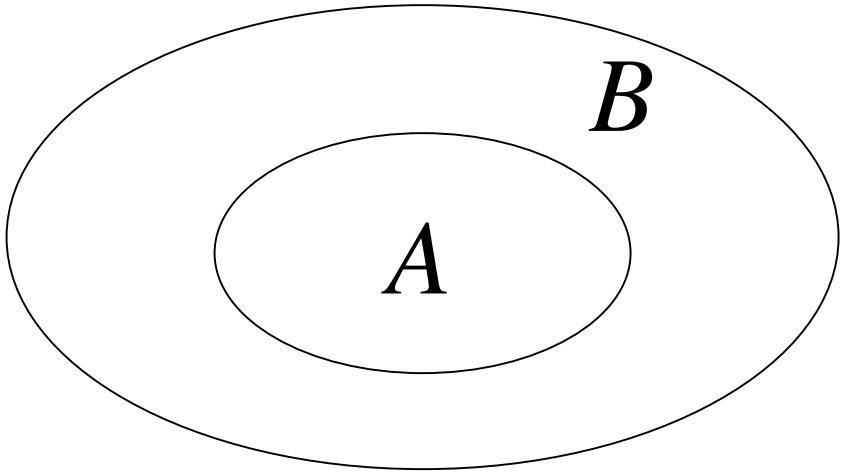
Reducibility

Problem A is reduced to problem B



If we can solve problem B then we can solve problem A



Problem A is reduced to problem B

If B is decidable then A is decidable

If A is undecidable then B is undecidable

Example: the halting problem

is reduced to

the state-entry problem

The state-entry problem

Inputs:

- Turing Machine M
- State q
- String w

Question: Does M enter state q on input w ?

Theorem:

The state-entry problem is undecidable

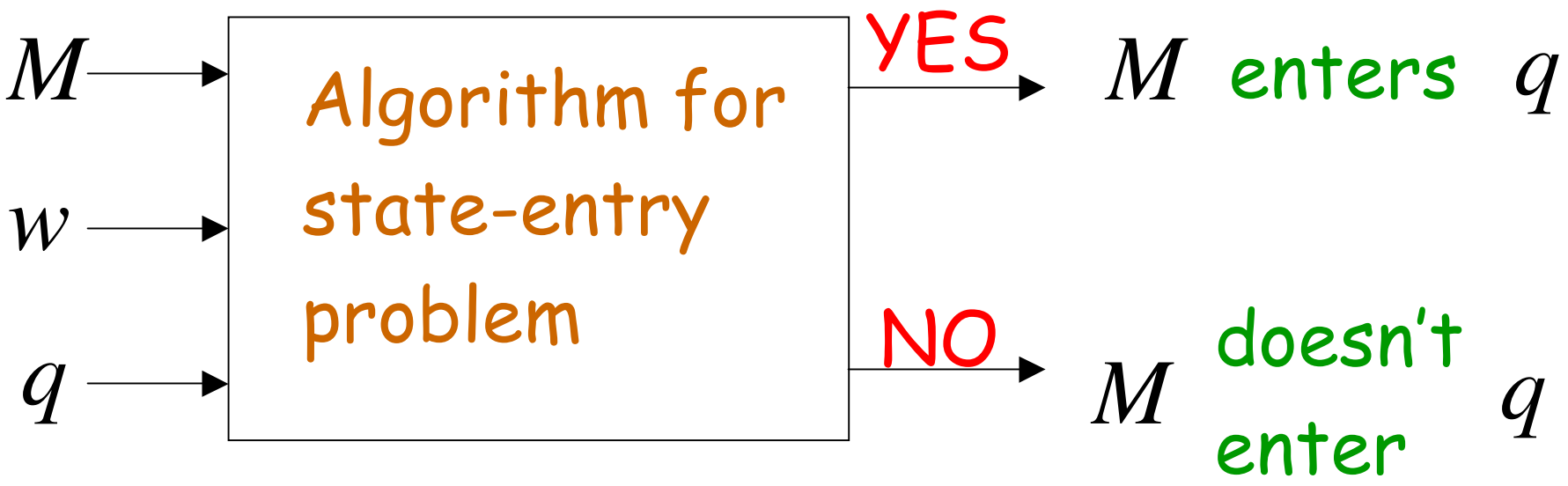
Proof: Reduce the halting problem to

the state-entry problem

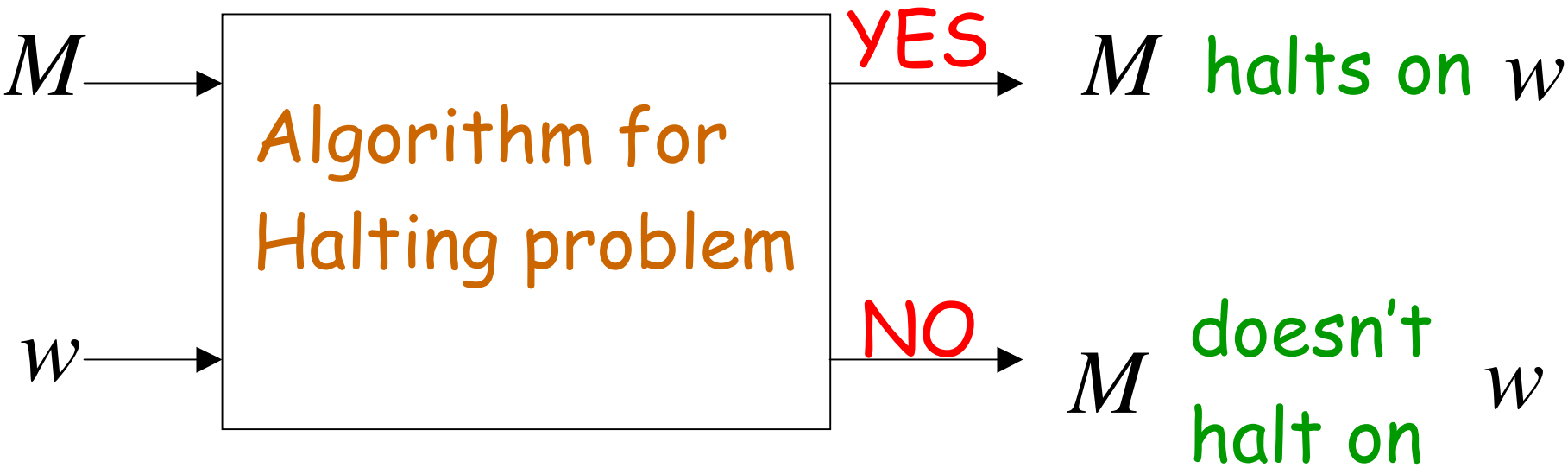
Suppose we have an algorithm (Turing Machine)
that solves the state-entry problem

We will construct an algorithm
that solves the halting problem

Assume we have the state-entry algorithm:

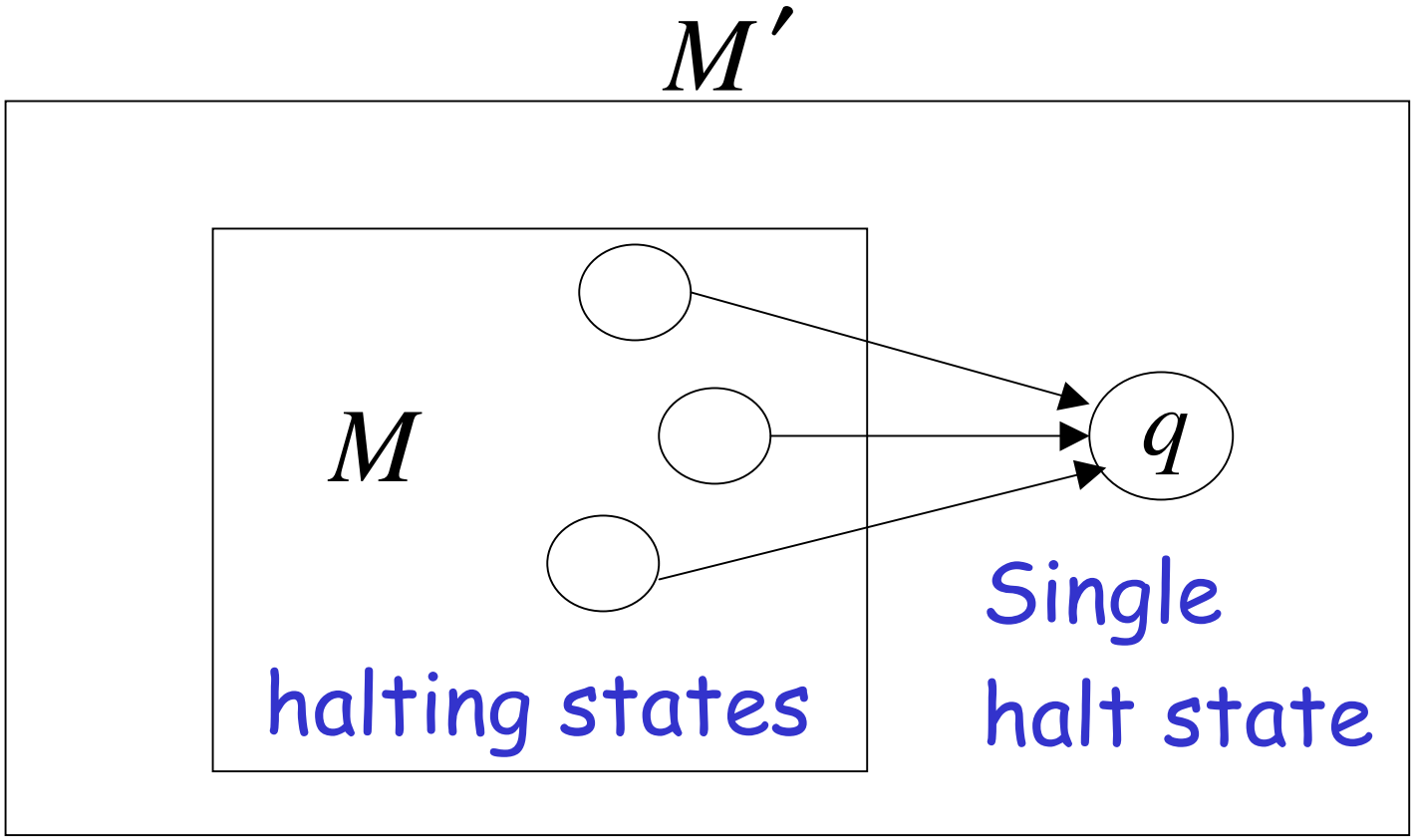


We want to design the halting algorithm:

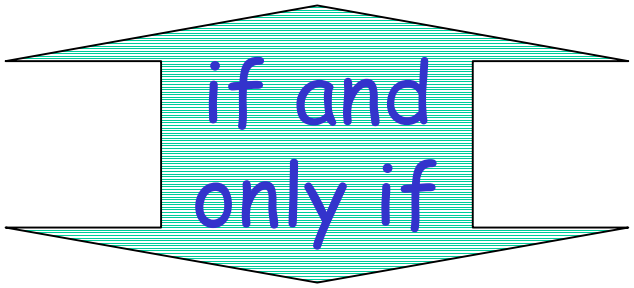


Modify input machine M :

- Add new state q
- From any halting state add transitions to q



M halts



M' halts on state q

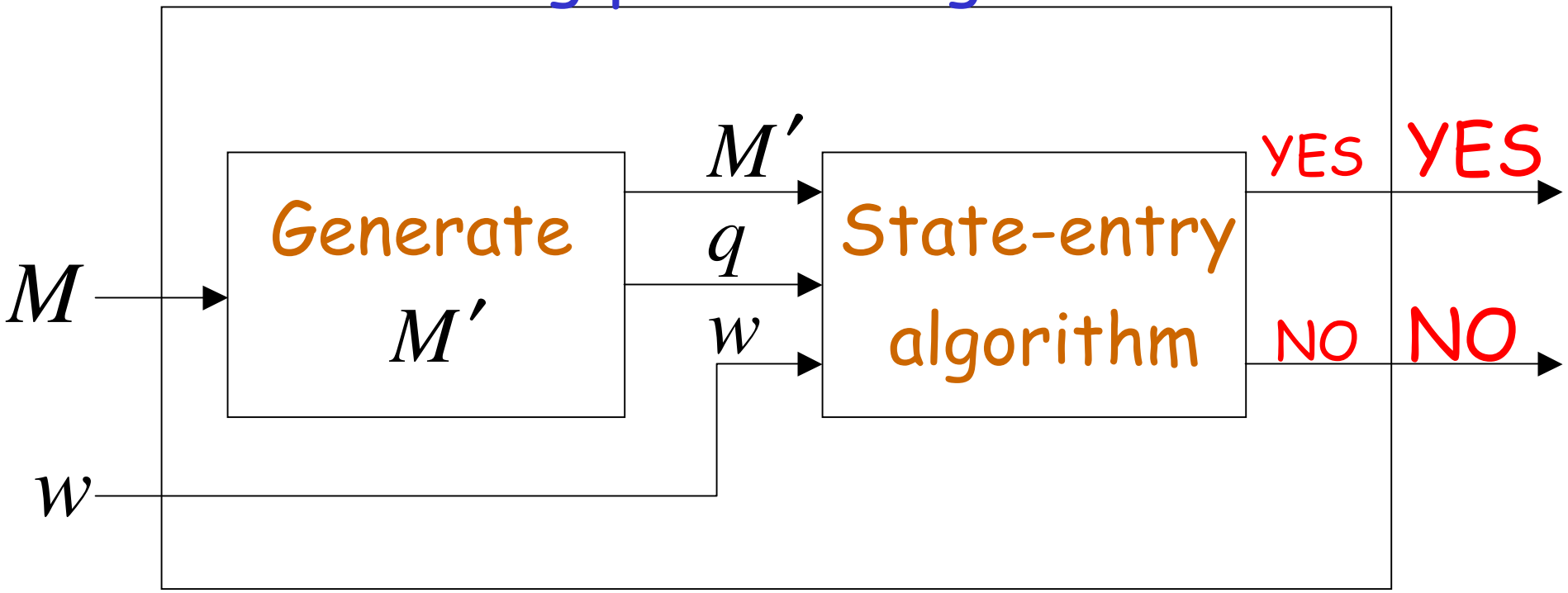
Algorithm for halting problem:

Inputs: machine M and string w

1. Construct machine M' with state q

2. Run algorithm for state-entry problem with inputs: M' , q , w

Halting problem algorithm



We reduced the halting problem
to the state-entry problem

Since the halting problem is undecidable,
it must be that the state-entry problem
is also undecidable

END OF PROOF

Another example:

the halting problem

is reduced to

the blank-tape halting problem

The blank-tape halting problem

Input: Turing Machine M

Question: Does M halt when started with a blank tape?

Theorem:

The blank-tape halting problem is undecidable

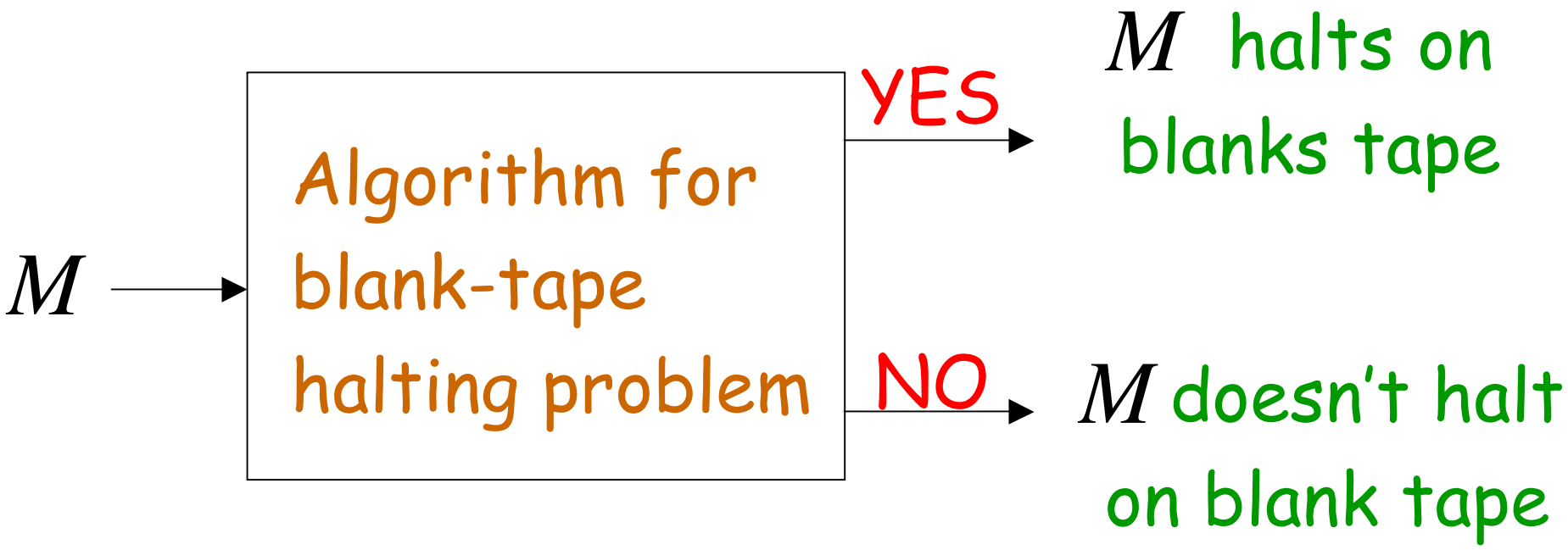
Proof: Reduce the halting problem to the

blank-tape halting problem

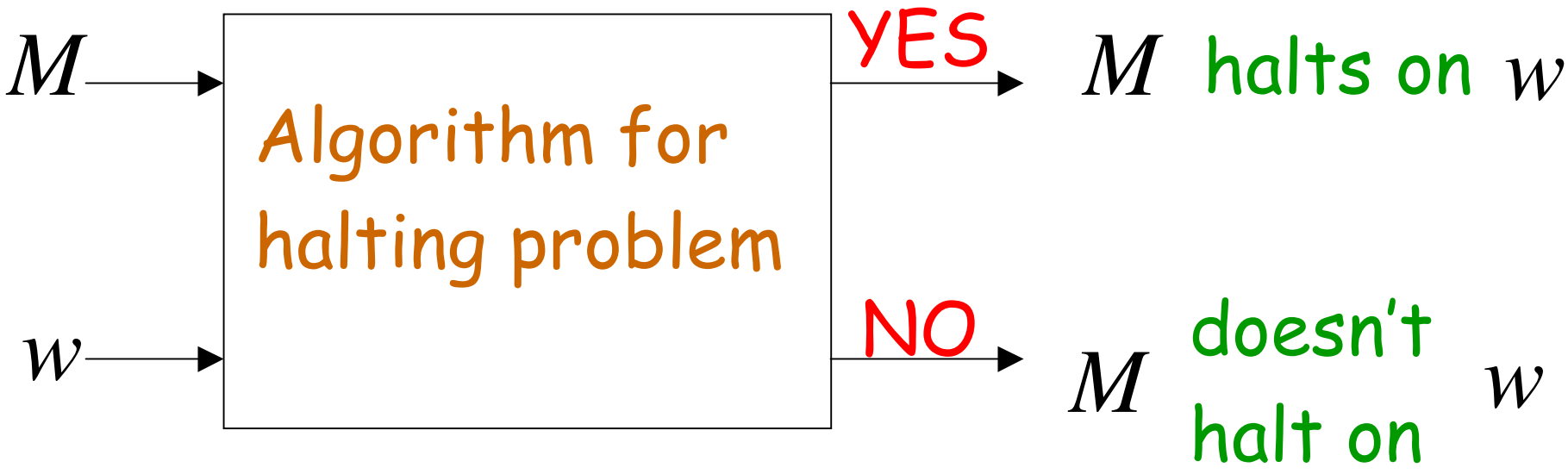
Suppose we have an algorithm
for the blank-tape halting problem

We will construct an algorithm
for the halting problem

Assume we have the
blank-tape halting algorithm:

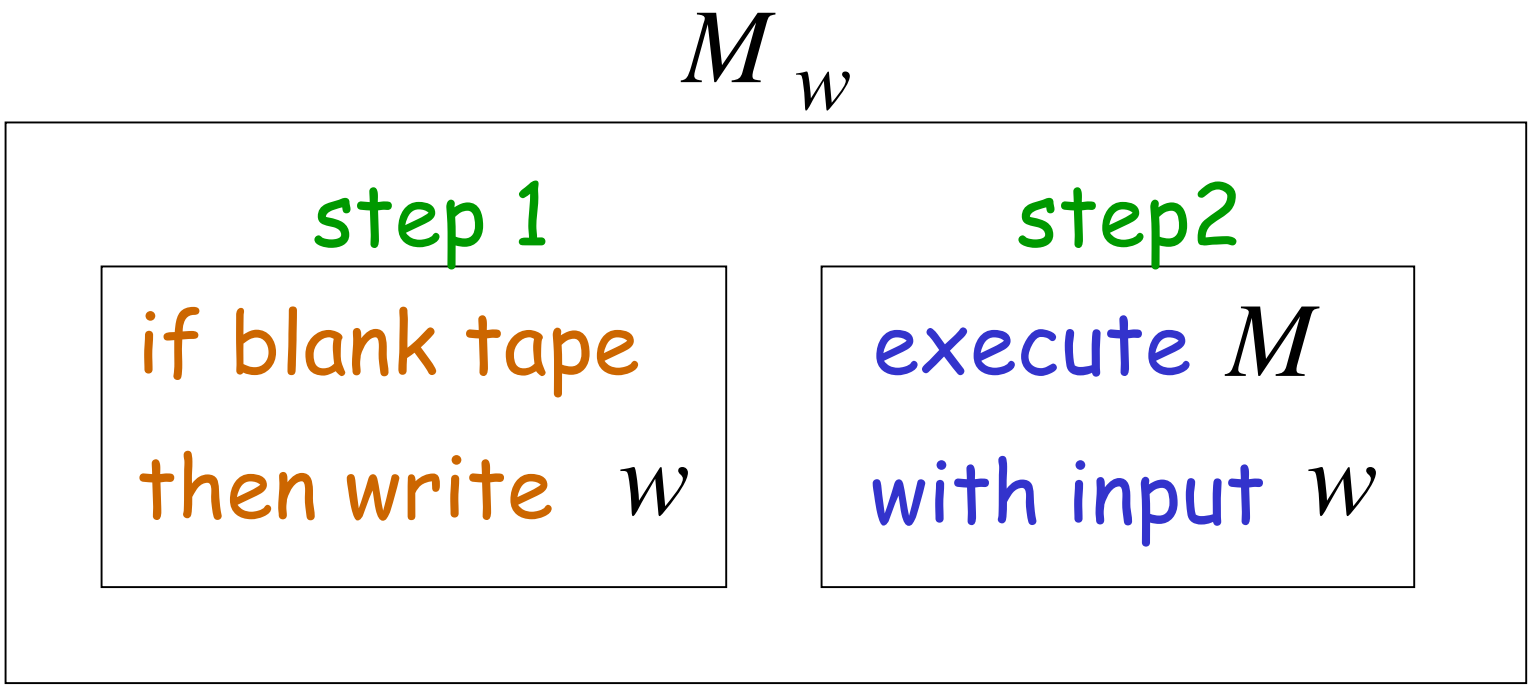


We want to design the halting algorithm:

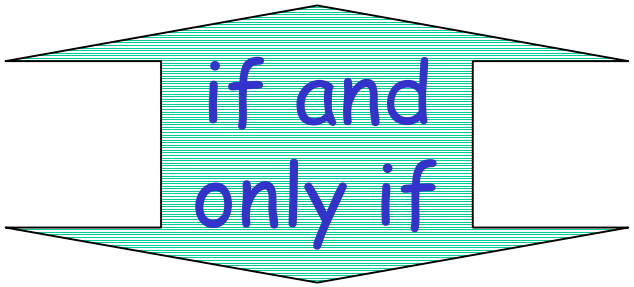


Construct a new machine M_w

- On blank tape writes w
- Then continues execution like M



M halts on input string w



M_w halts when started with blank tape

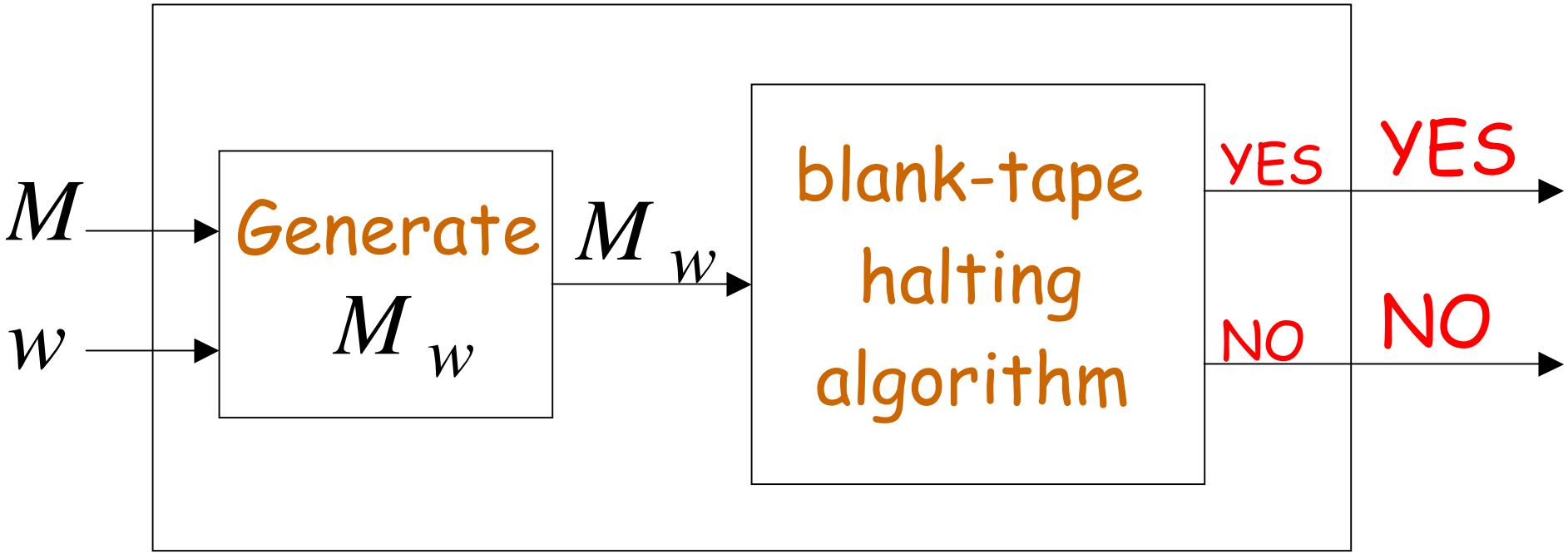
Algorithm for halting problem:

Inputs: machine M and string w

1. Construct M_w

2. Run algorithm for
blank-tape halting problem
with input M_w

Halting problem algorithm



We reduced the halting problem
to the blank-tape halting problem

Since the halting problem is undecidable,
the blank-tape halting problem is
also undecidable

END OF PROOF

Summary of Undecidable Problems

Halting Problem:

Does machine M halt on input w?

Membership problem:

Does machine M accept string w?

In other words: Is a string w member of a recursively enumerable language L?

Blank-tape halting problem:

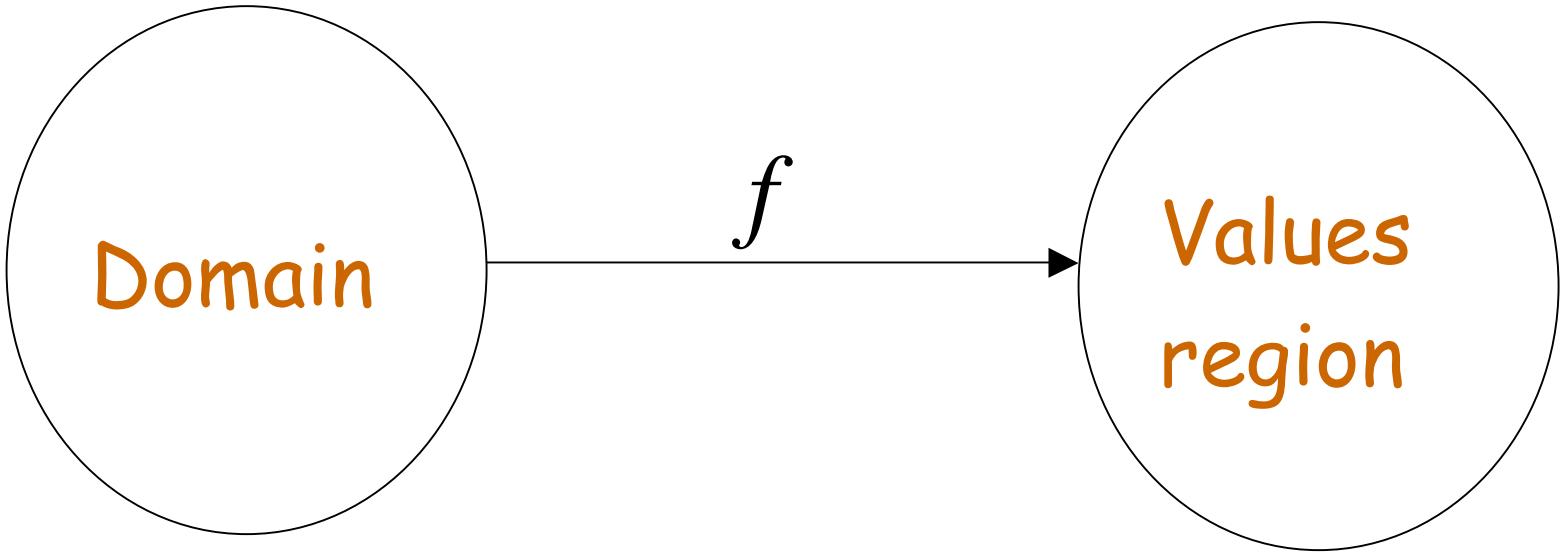
Does machine M halt when starting on blank tape?

State-entry Problem:

Does machine M enter state q on input w ?

Uncomputable Functions

Uncomputable Functions



A function is uncomputable if it cannot be computed for all of its domain

An uncomputable function:

$$f(n) = \begin{cases} \text{maximum number of moves until} \\ \text{any Turing machine with } n \text{ states} \\ \text{halts when started with the blank tape} \end{cases}$$

Theorem: Function $f(n)$ is uncomputable

Proof: Assume for contradiction that $f(n)$ is computable

Then the blank-tape halting problem is decidable

Algorithm for blank-tape halting problem:

Input: machine M

1. Count states of M : m
2. Compute $f(m)$
3. Simulate M for $f(m)$ steps starting with empty tape

If M halts then return YES

otherwise return NO

Therefore, the blank-tape halting problem is decidable

However, the blank-tape halting problem is undecidable

Contradiction!!!

Therefore, function $f(n)$ in uncomputable

END OF PROOF

Rice's Theorem

Definition:

Non-trivial properties of
recursively enumerable languages:

any property possessed by some (not all)
recursively enumerable languages

Some non-trivial properties of recursively enumerable languages:

- L is empty
- L is finite
- L contains two different strings of the same length

Rice's Theorem:

Any non-trivial property of
a recursively enumerable language
is undecidable

We will prove some non-trivial properties without using Rice's theorem

Theorem:

For any recursively enumerable language L it is undecidable to determine whether L is empty

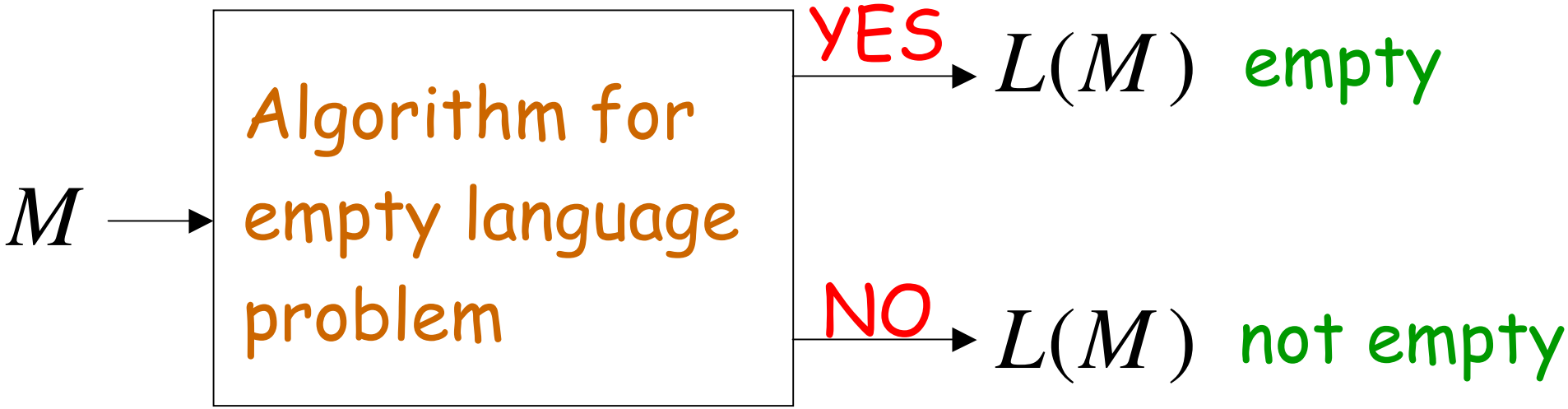
Proof:

We will reduce the membership problem to this problem

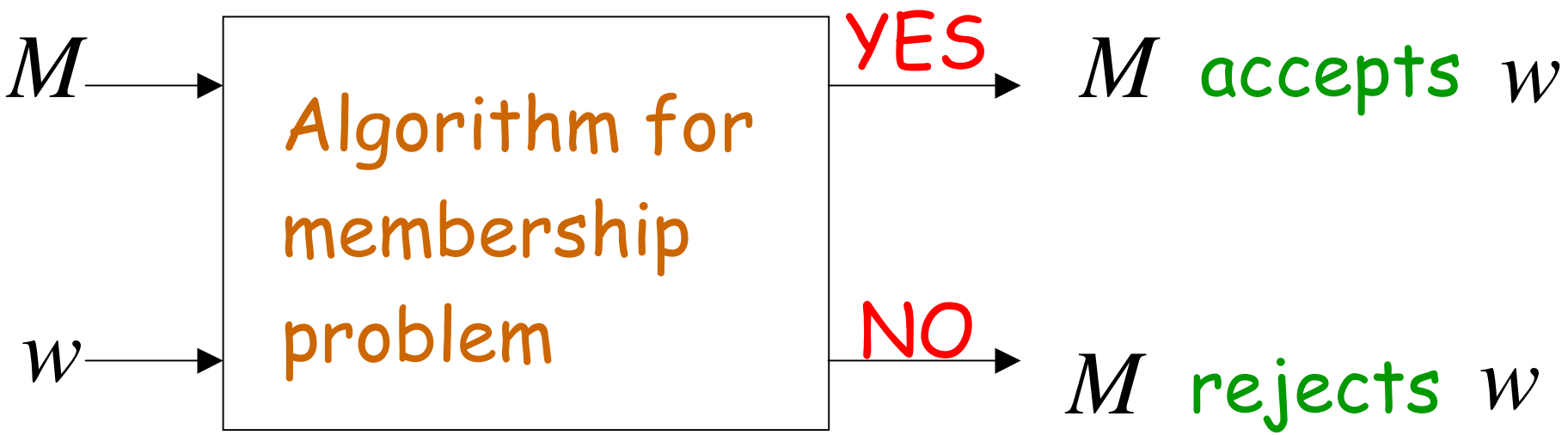
Let M be the machine that accepts L

$$L(M) = L$$

Assume we have the empty language algorithm:



We will design the membership algorithm:

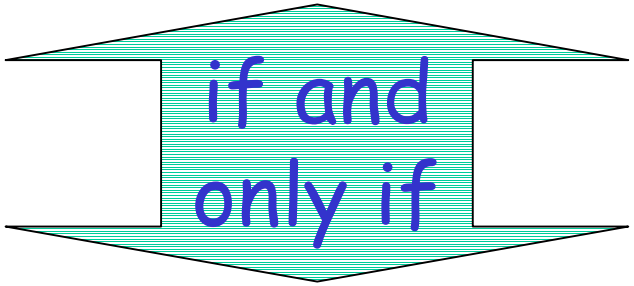


First construct machine M_w:

When M enters a final state,
compare original input string with w

Accept if original input is
the same with w

$$w \in L$$



$L(M_w)$ is not empty

$$L(M_w) = \{w\}$$

Algorithm for membership problem:

Inputs: machine M and string w

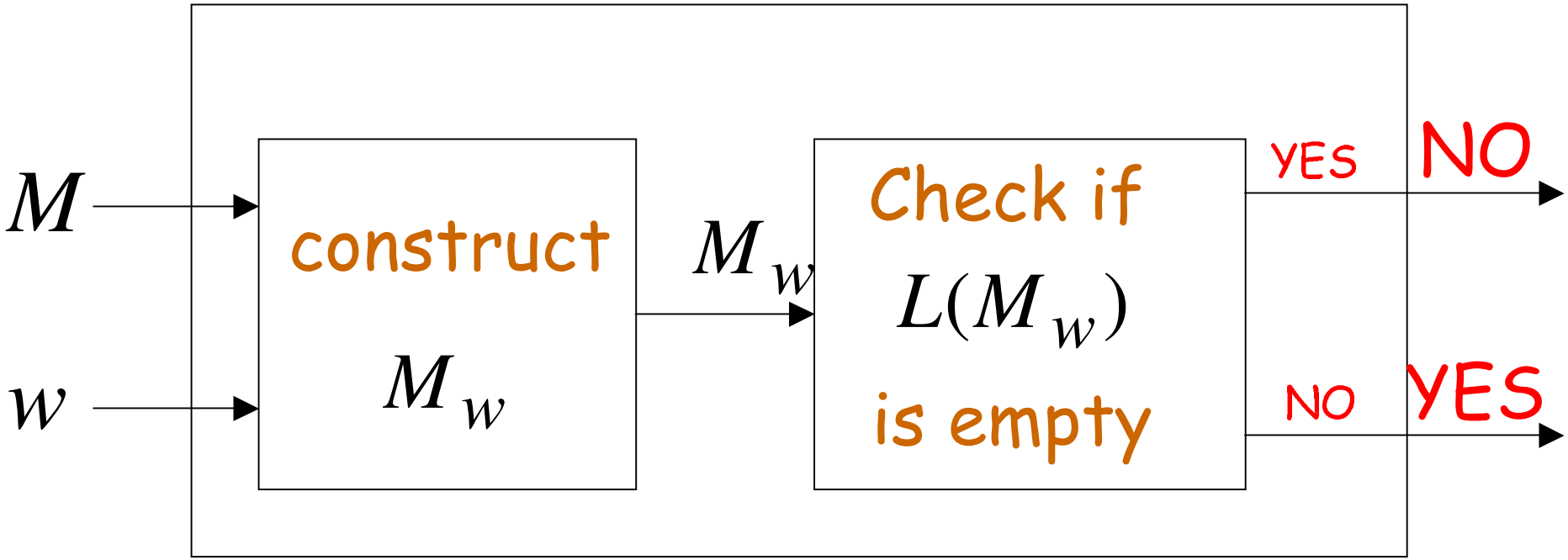
1. Construct M_w

2. Determine if $L(M_w)$ is empty

YES: then $w \notin L(M)$

NO: then $w \in L(M)$

Membership algorithm



END OF PROOF