

DVA337

FABER

Formal Languages, Automata and Models of Computation

Lecture 12

Mälardalen University

2015

Content

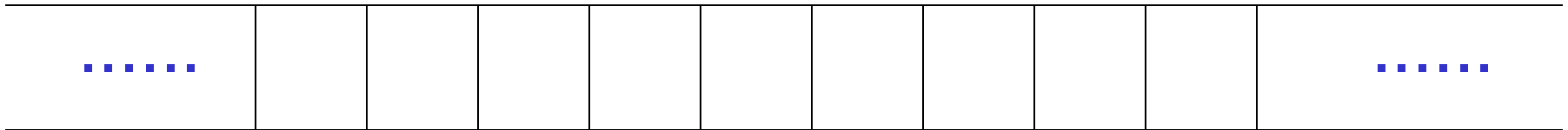
Universal Turing Machine
Chomsky Hierarchy
Decidability
Reducibility
Uncomputable Functions
Rice's Theorem
Church-Turing Thesis

TURING MACHINES

Based on C Busch, RPI, Models of Computation

Standard Turing Machine

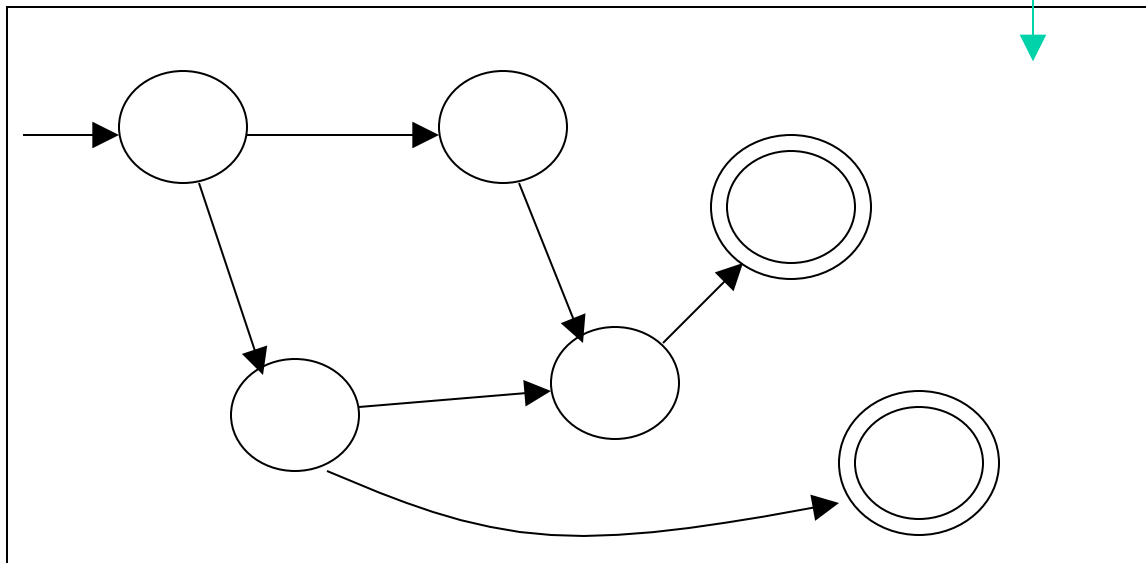
Tape



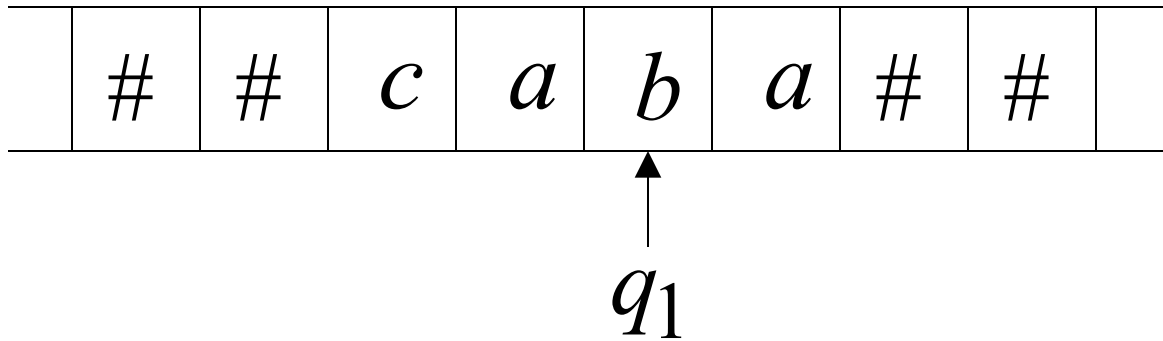
Read-Write head



Control Unit

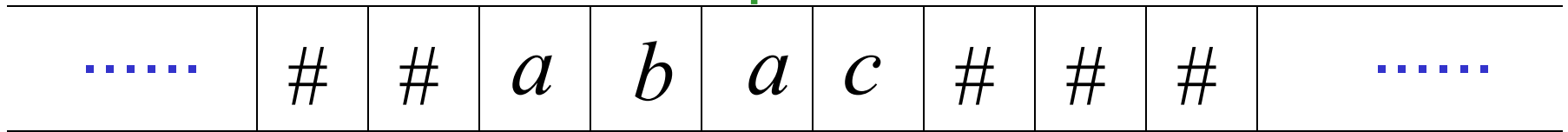


Configuration



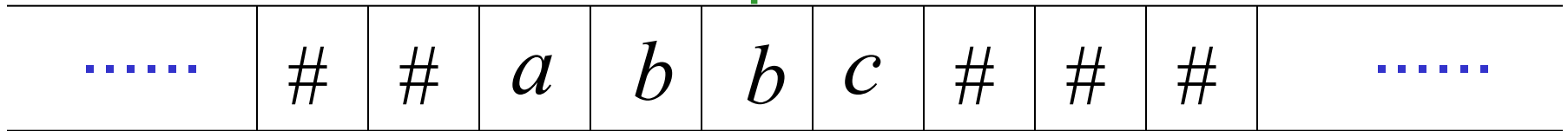
Instantaneous description: $ca\ q_1\ ba$

Step 1

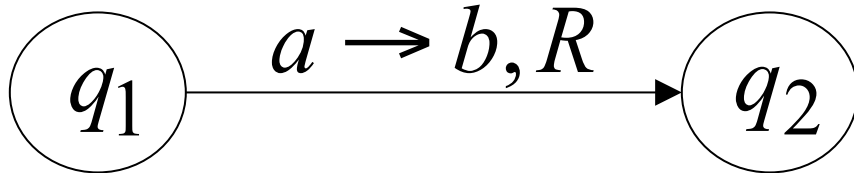


q_1

Step 2

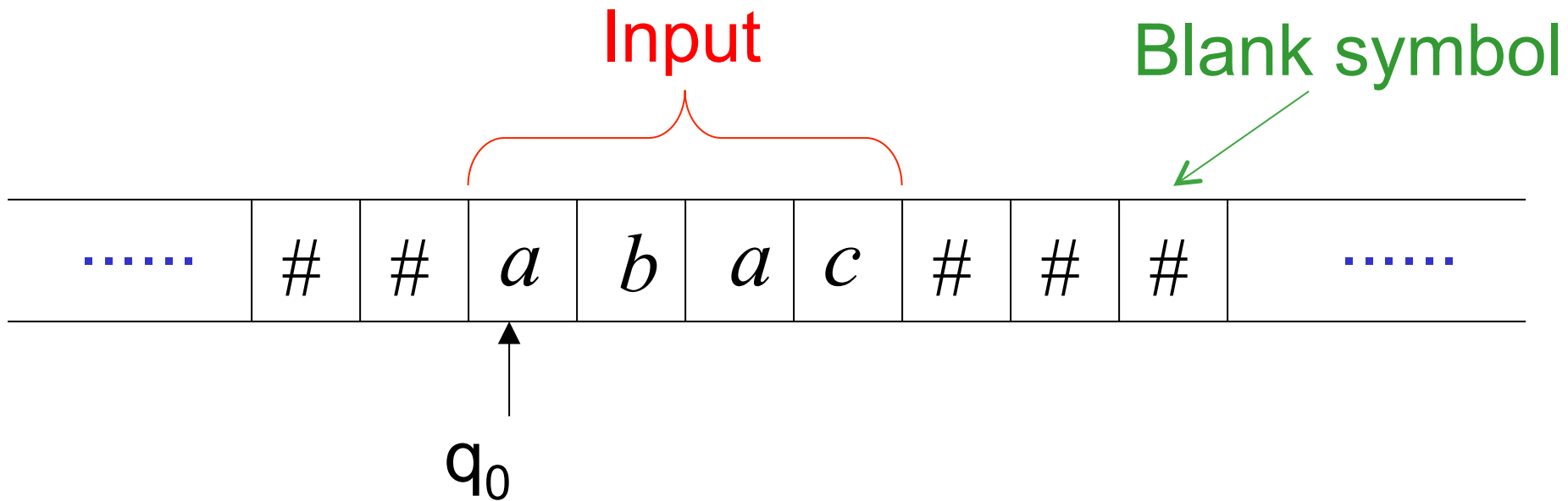


q_2



$$\delta(q_1, a) = (q_2, b, R)$$

$$abq_1ac \mapsto abbq_2c$$



Head starts at the leftmost position of the input

$q_0 w \mapsto \dots$, w input, $w = abac$ in the example above

The Accepted Language

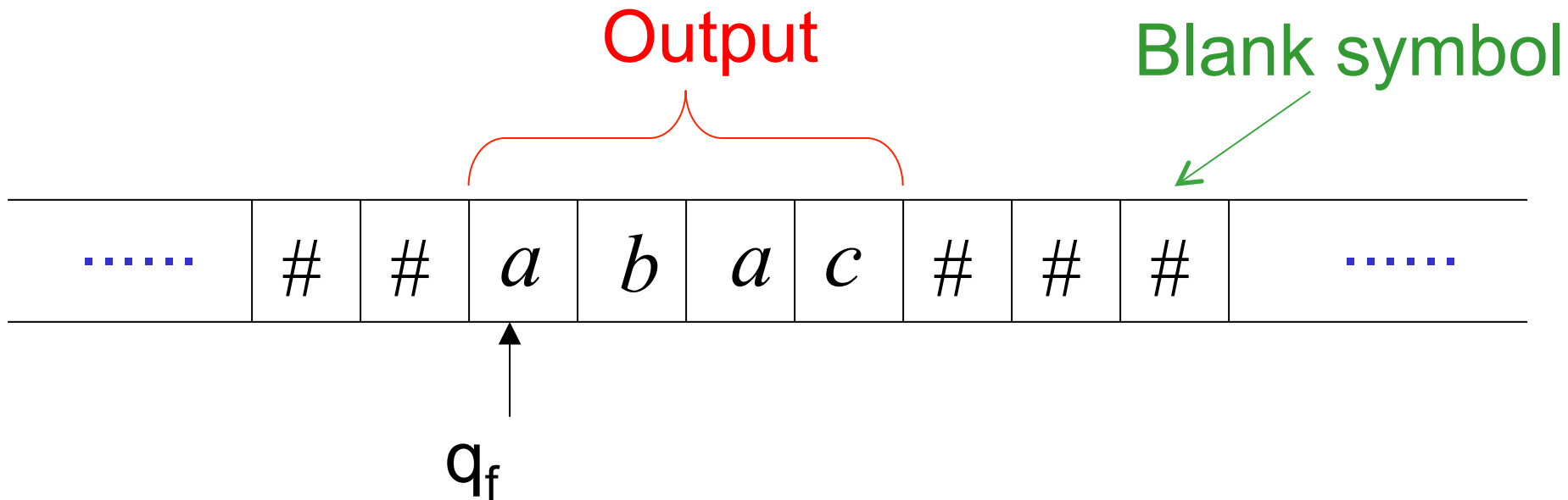
For any Turing Machine M Any x_1 and x_2

$$L(M) = \{w : q_0 w \xrightarrow{*} x_1 q_f x_2\}$$

The diagram illustrates the definition of the accepted language $L(M)$ for a Turing Machine M . The equation is $L(M) = \{w : q_0 w \xrightarrow{*} x_1 q_f x_2\}$. Green arrows point from the text 'Initial state' to q_0 , 'Final state' to q_f , and 'Any x_1 and x_2 ' to x_1 and x_2 .

Initial state

Final state

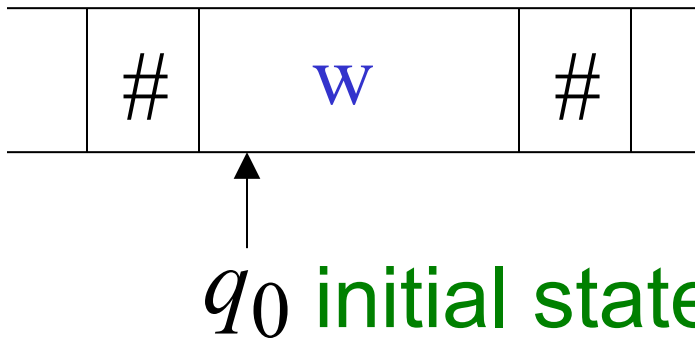


If head stops in a final state as at the beginning of string surrounded by blanks
 $\dots \mapsto q_f w$, $w = abac$ in the example above

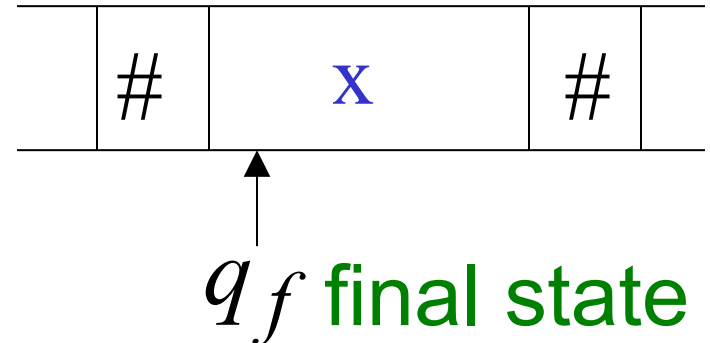
Definition: turing machine as a function

A Turing machine computes a function $f : A \rightarrow B$ if for all $w \in A$, with $x = f(w)$, $q_0 w \mapsto^* q_f x$

Initial configuration



Final configuration



Recursive and Recursively Enumerable Languages

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

- Halt and accept the input; (halt in final)
- Halt and reject the input; or (halt in non-final)
- Never halt.

Recursively Enumerable Languages (semi decidable)

A language is **recursively enumerable** if there exists a Turing machine that accepts every string of the language, and does not accept strings that are not in the language.

Strings that are not in the language may be rejected or may cause the Turing machine to go into an infinite loop; semi-decidable.

Recursively Enumerable Languages

Given such a semi-decision procedure how can you construct an enumeration procedure (Turing machine)?

Recursive Languages (decidable)

A language is **recursive** if there exists a Turing machine that **accepts every string** of the language **and rejects every string** (over the same alphabet) that is not in the language.

If a language L is recursive, then its complement is recursive.

As we get the answer YES/NO, **recursive languages are decidable.**

Non-Recursively Enumerable Languages

A language is non-recursively enumerable for which there exists no Turing machine.

Do they exist?

Recursively Enumerable Languages

Unrestricted Grammars

Productions

$$u \rightarrow v$$

String of variables
and terminals



String of variables
and terminals



Example of unrestricted grammar

$$S \rightarrow aBc$$

$$aB \rightarrow cA$$

$$Ac \rightarrow d$$

Theorem

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

Why this name?

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:

$$S \rightarrow abc \mid aAbc$$

$$Ab \rightarrow bA$$

$$Ac \rightarrow Bbcc$$

$$bB \rightarrow Bb$$

$$aB \rightarrow aa \mid aaA$$

How does
it work?

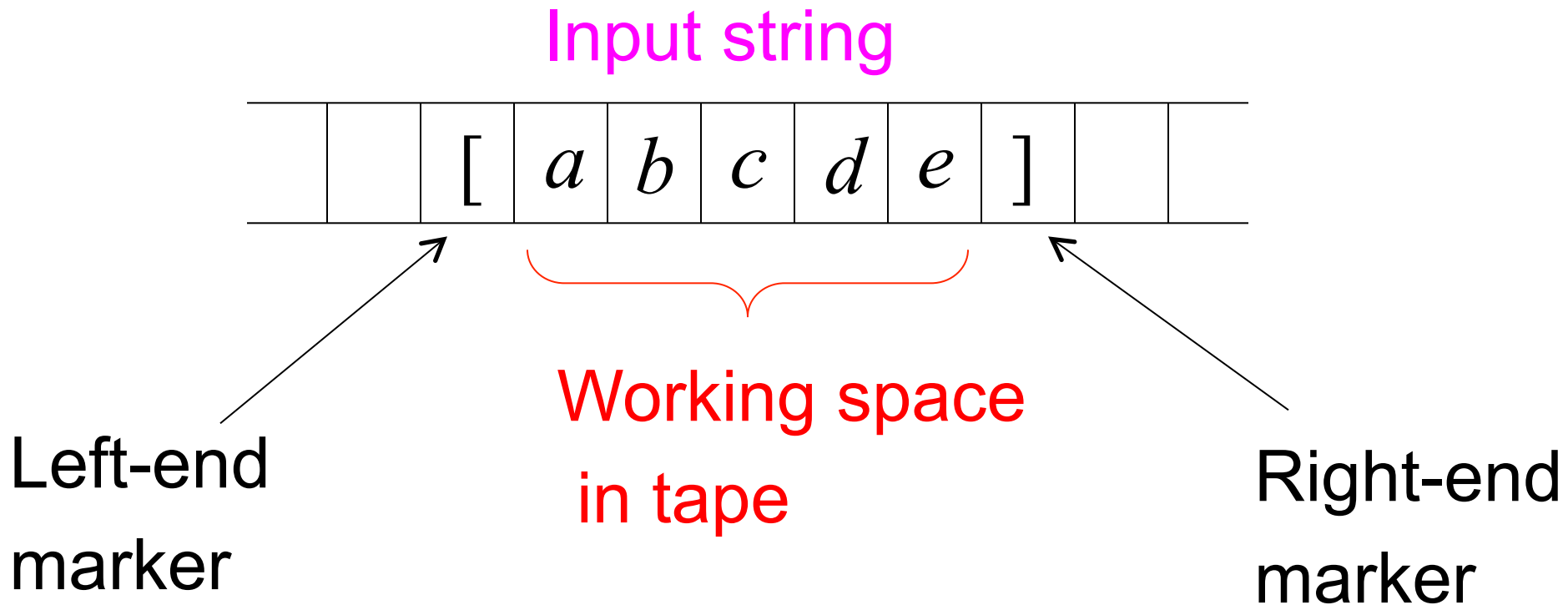
Theorem

A language L is context sensitive
if and only if
it is accepted by a Linear-Bounded
automaton.

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)



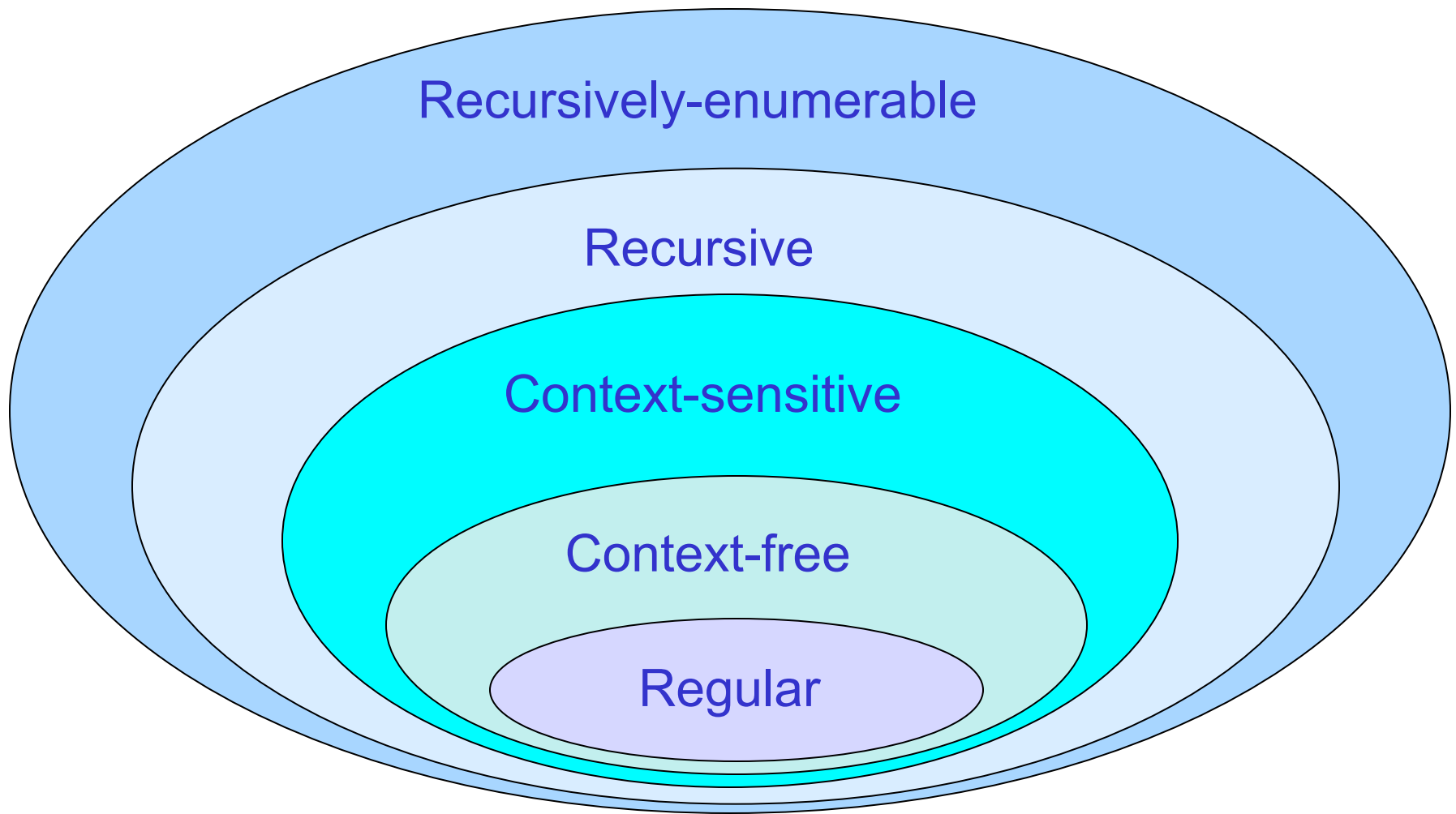
All computation is done between end markers.

Observation

1. Every context-sensitive language is recursive
(trivially by the existence of the LBA)
2. Not every recursive language is
context-sensitive
(maybe not that surprising – LBAs have finite
finite memory, while TMs have infinite)

The Chomsky Hierarchy

The Chomsky Language Hierarchy



Non-recursively enumerable

Universal Turing Machine

Limitation of Standard Turing Machines:

Turing Machines are “hardwired”



they execute
only one program

Better are reprogrammable machines.

Solution: Universal Turing Machine

Characteristics:

- Reprogrammable machine
- Simulates any other Turing Machine

Universal Turing Machine

simulates any other Turing Machine M

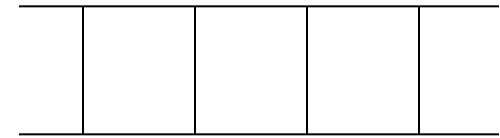
Input of Universal Turing Machine

- Description of transitions of M
- Initial tape contents of M

Three tapes

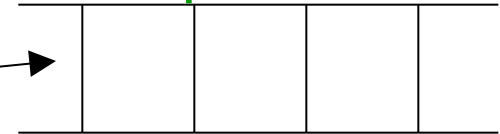


Tape 1



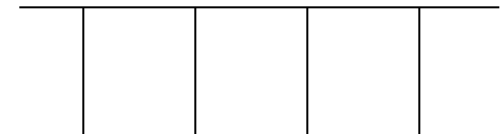
Description of M

Tape 2



Tape Contents of M

Tape 3



States of M

Tape 1





--	--	--	--	--

Description of M

We encode/describe Turing Machine M
as a string of symbols.

Tape Contents Encoding

Alphabet Encoding

Symbols:	a	b	c	d	...
					
Encoding:	1	11	111	1111	

State Encoding

States: q_1 q_2 q_3 q_4 \dots



Encoding: 1 11 111 1111

Head Move Encoding

Move: L R



Encoding: 1 11

Transition Encoding

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

Encoding:

10101101101
separator

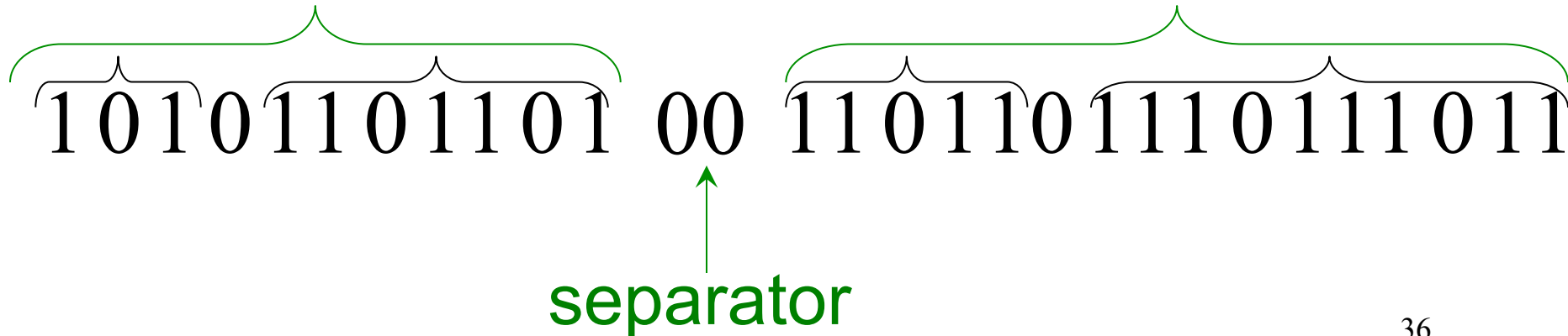
Machine Encoding

Transitions:

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

$$\delta(q_2, b) = (q_3, c, R)$$

Encoding:



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, (Turing Machine 3)
..... }

.....

Decidability

Consider problems with answer YES or NO.

Examples

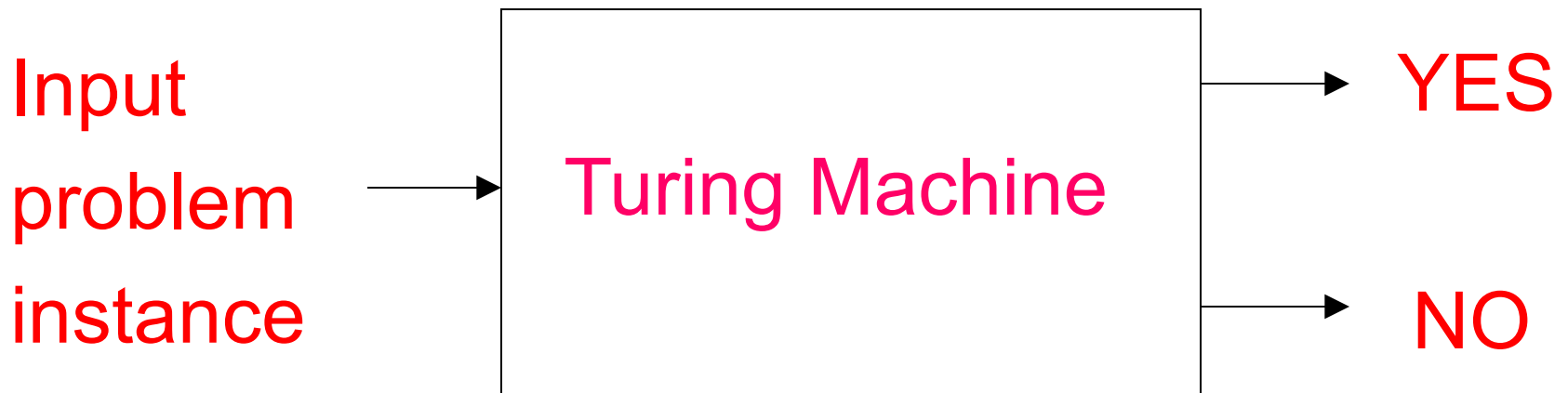
- Does Machine M have three states ?
- Is string w a binary number?
- Does DFA M accept any input?

A problem is decidable if some Turing machine solves (decides) the problem.

Decidable problems:

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

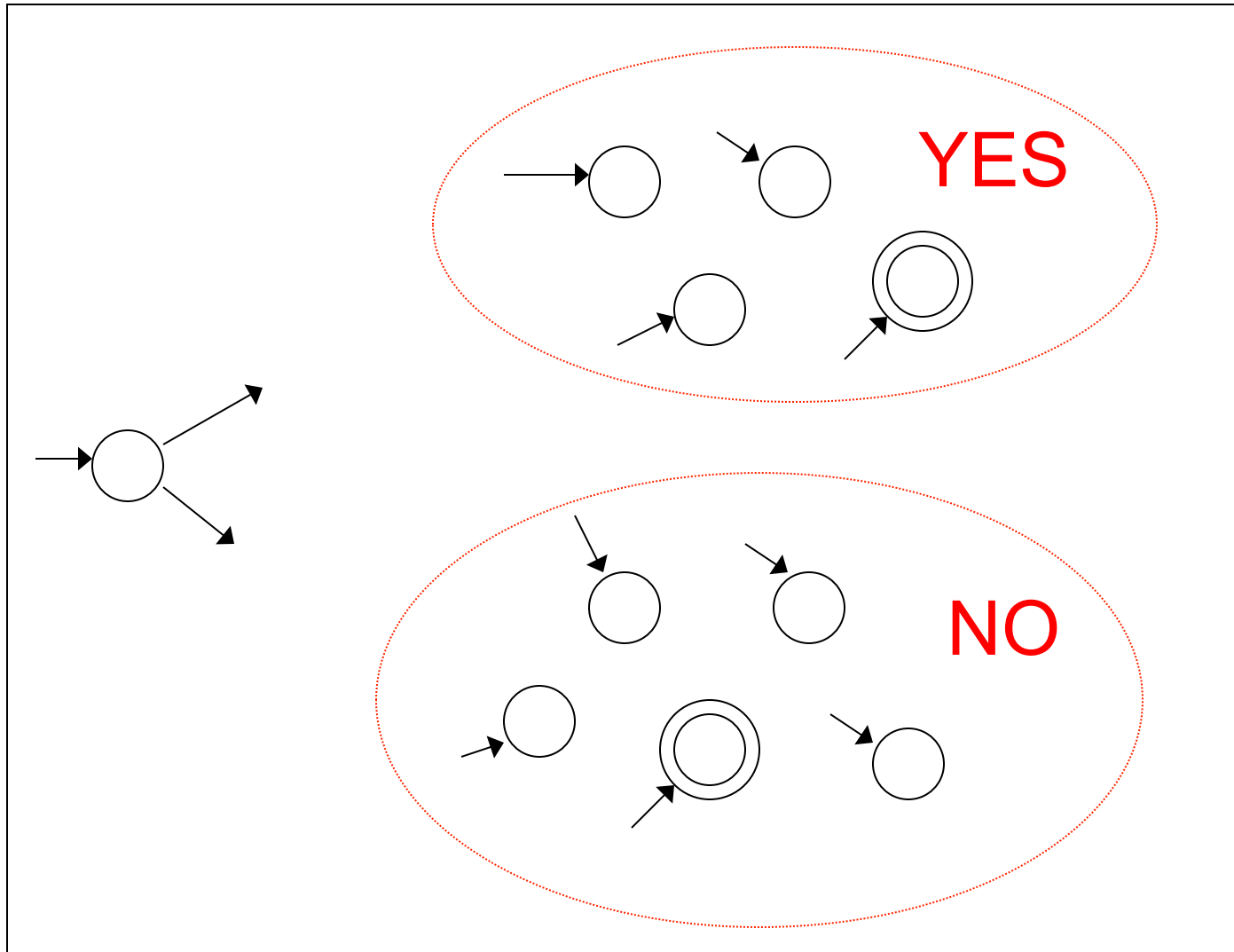
The Turing machine that solves a problem answers **YES** or **NO** for each instance.



The machine that ***decides*** a problem:

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

Turing Machine that decides a problem



YES and NO states are halting states

Some problems are undecidable:

There is no Turing Machine that
solves **all instances** of the problem.

A famous undecidable problem:

The halting problem

The Halting Problem

Input: • (encoding of) Turing Machine M
• String w

Question: Does M halt on w ?

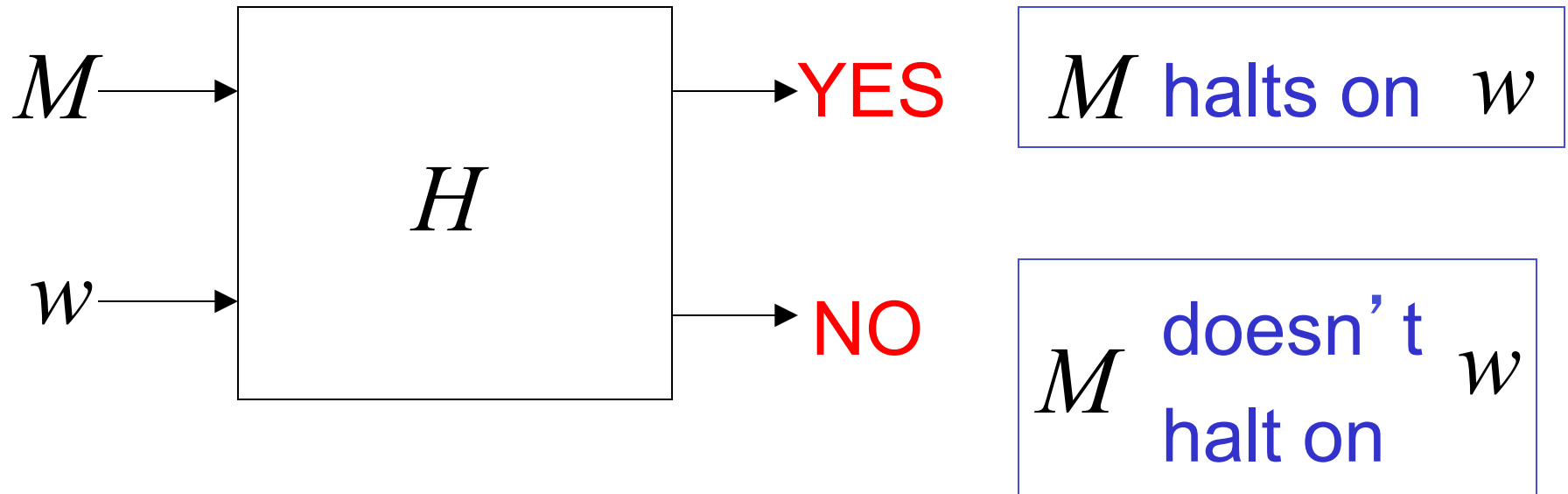
Theorem

The halting problem is **undecidable**.

Proof

Assume to the contrary that
the halting problem is decidable.

There exists Turing Machine H
that solves the halting problem



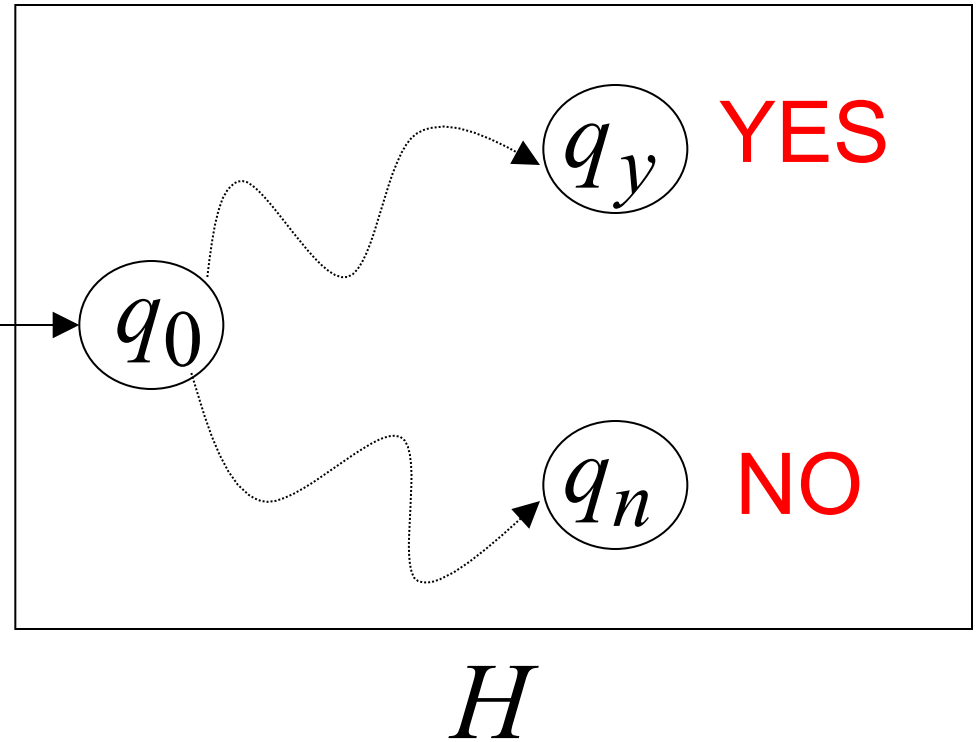
Construction of H

Input:
initial tape contents

w_M w

Encoding
of M

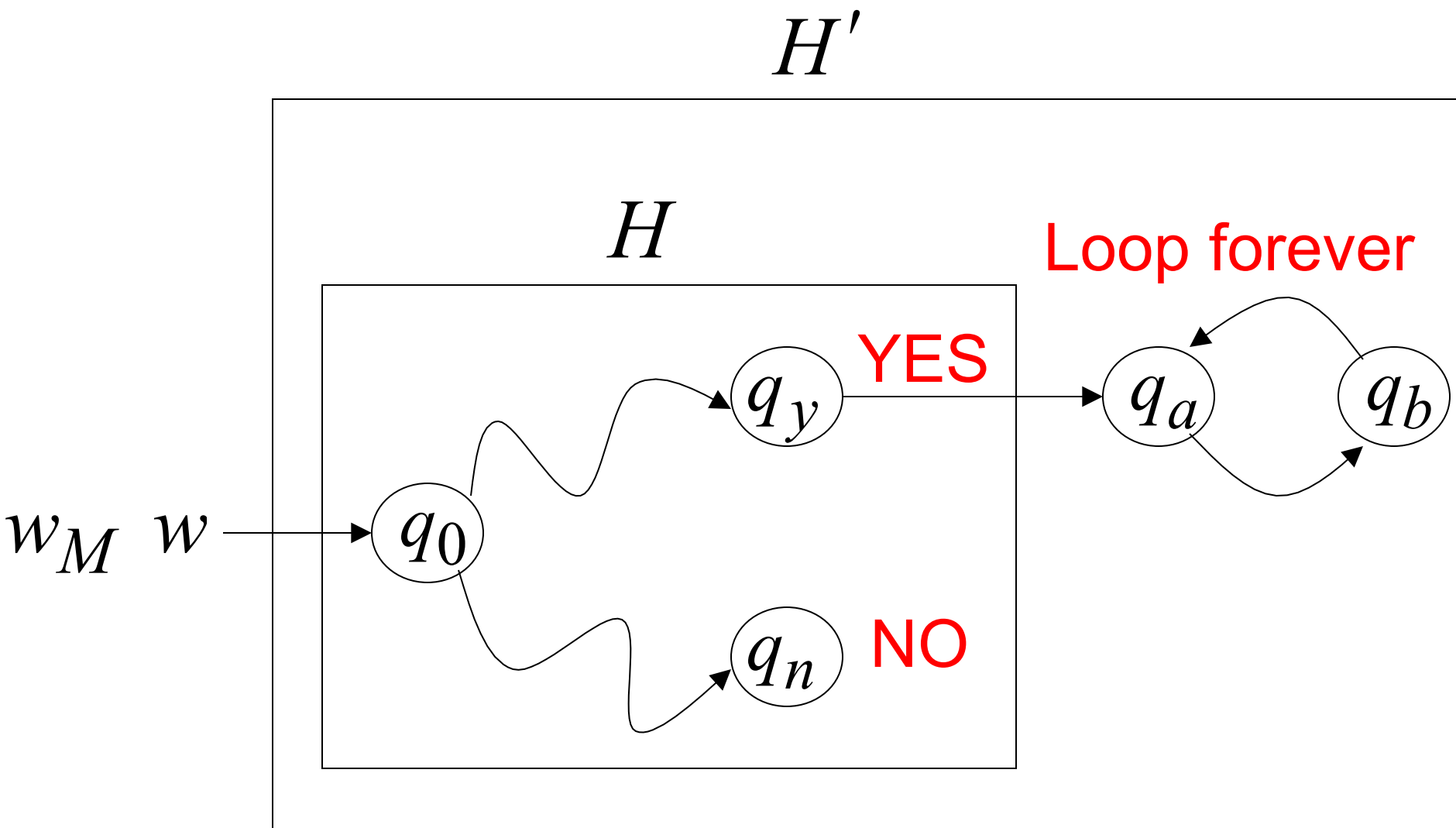
String
 w



Construct machine H'

If H returns YES then loop forever.

If H returns NO then halt.



Construct machine \hat{H}

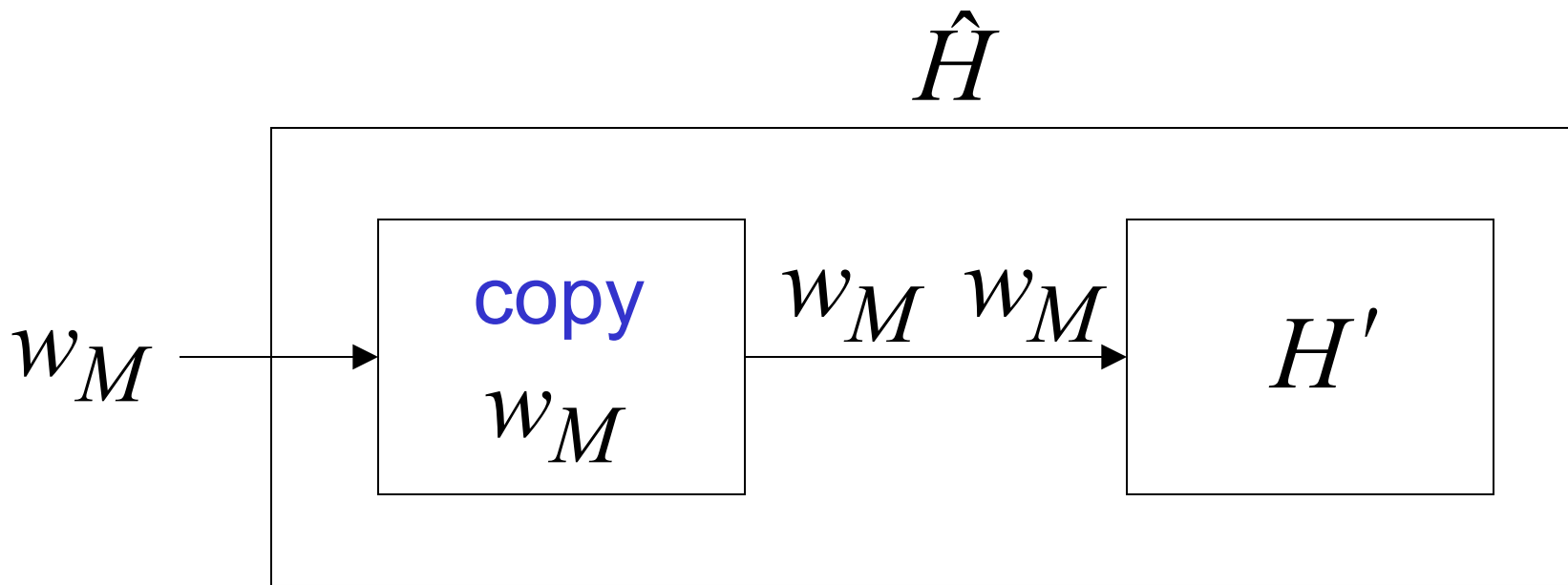
Input: w_M (machine M)

Itself!

If M halts on input w_M

Then loop forever

Else halt



Run machine \hat{H} with input itself

Input: $w_{\hat{H}}$ (machine \hat{H})

If \hat{H} halts on input $w_{\hat{H}}$

Then loop forever

Else halt

\hat{H} on input $w_{\hat{H}}$:

If \hat{H} halts then loops forever.

If \hat{H} doesn't halt then it halts.

CONTRADICTION !

This means that

The halting problem is undecidable.

END OF PROOF

Another proof of the same theorem:

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

Recursive vs. Recursively Enumerable Languages

<http://www.cs.colostate.edu/~massey/Teaching/cs301/RestrictedAccess/Slides/301lecture23.pdf>

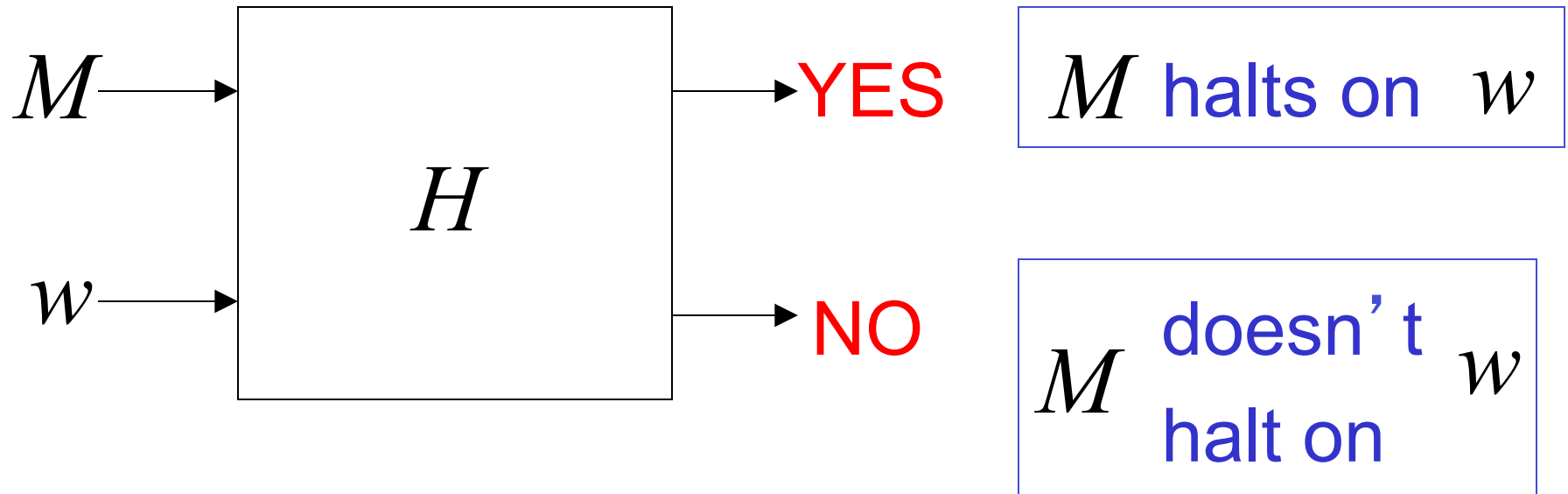
Theorem

The halting problem is undecidable.

Proof

Assume to the contrary that
the halting problem is decidable.

There exists Turing Machine H
that solves the halting problem.



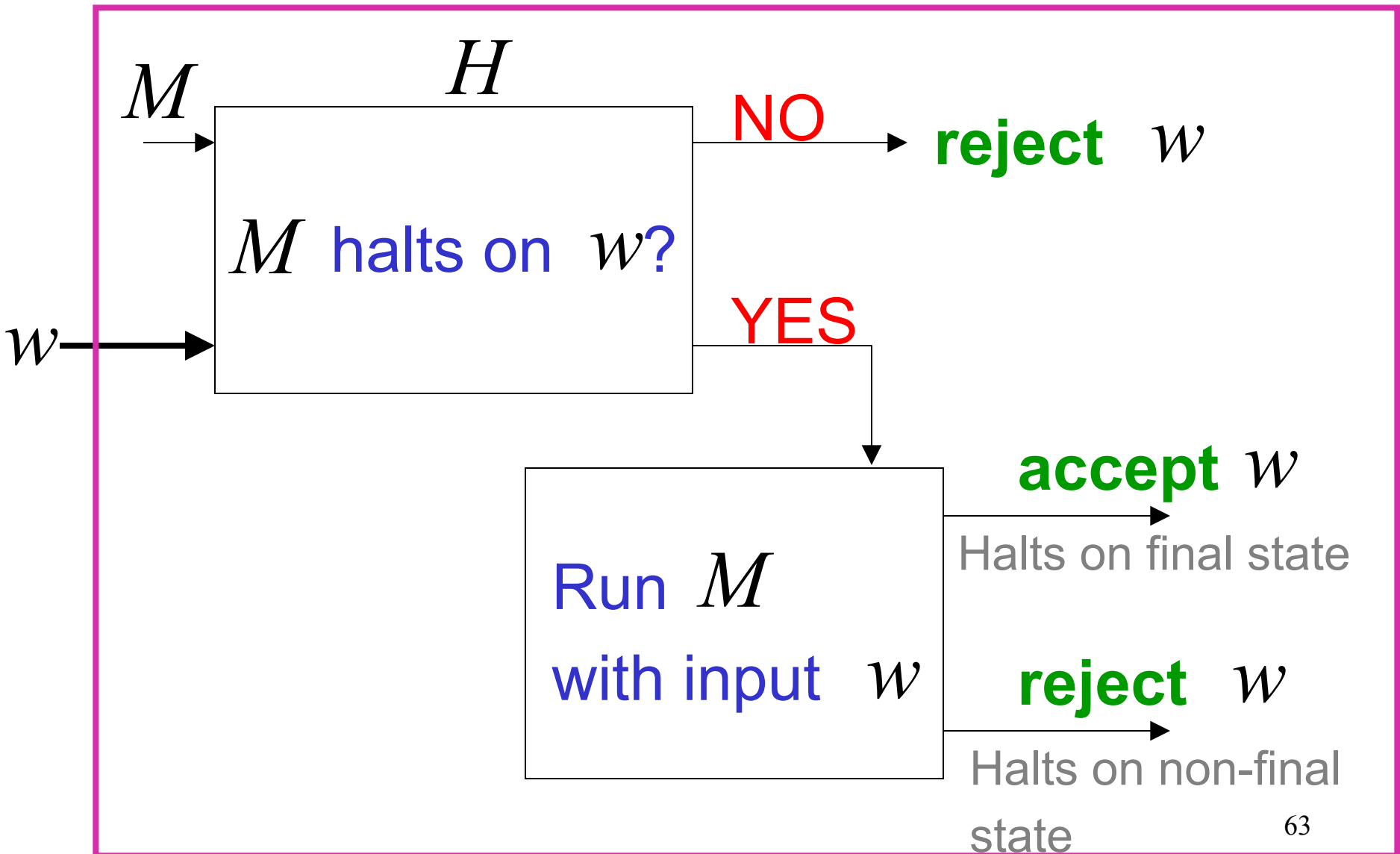
Let L be a recursively enumerable language.

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, we have proven that every recursively enumerable language is also recursive.

But there are recursively enumerable languages which are not recursive.

Contradiction!

Therefore, the halting problem is undecidable.

END OF PROOF

A simple undecidable problem:

The Membership Problem

The Membership Problem

Input: • Turing Machine M
• String w

Question: Does M accept w ?

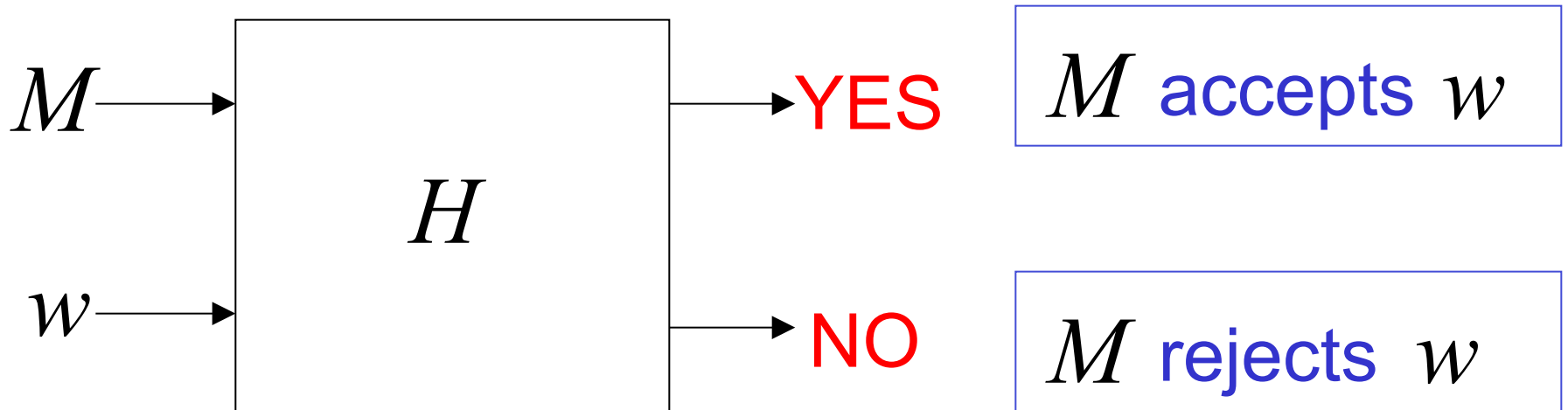
Theorem

The membership problem is undecidable.

Proof

Assume to the contrary that
the membership problem is decidable.

There exists a Turing Machine H
that solves the membership problem



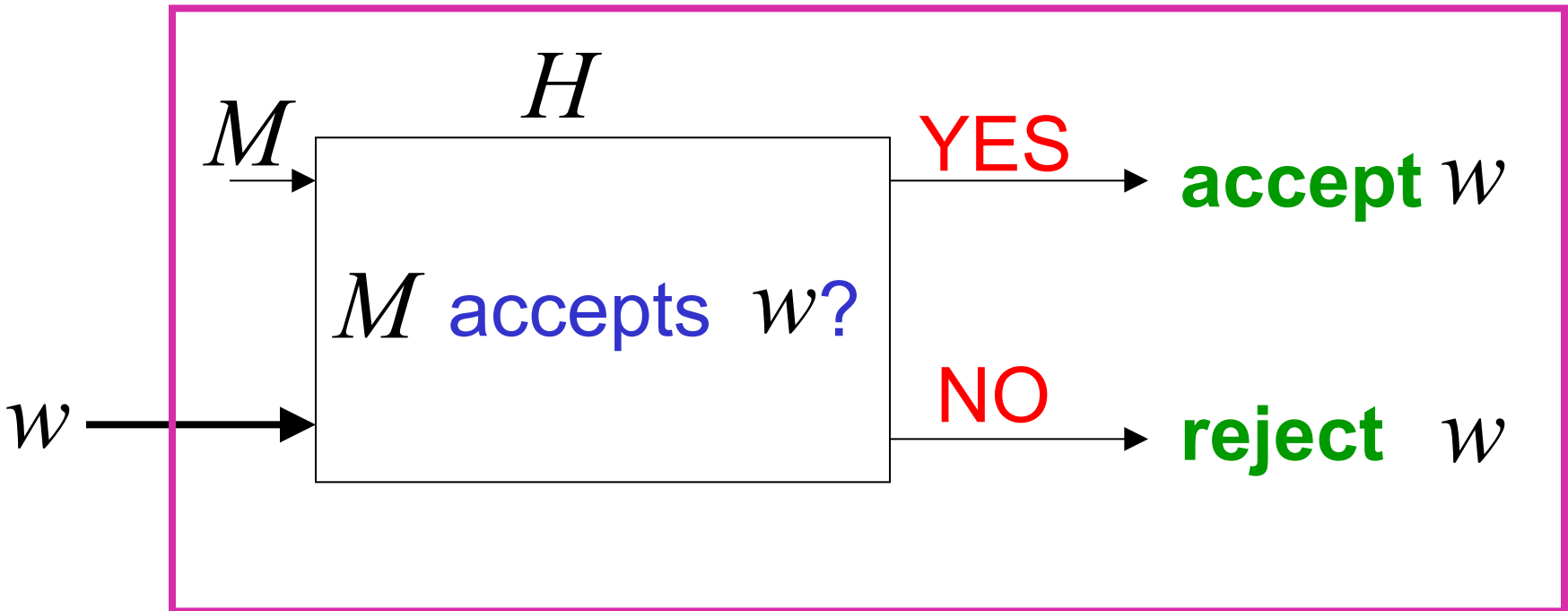
Let L be a recursively enumerable language.

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, we have proven that every recursively enumerable language is also recursive.

But there are recursively enumerable languages which are not recursive.

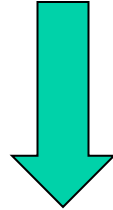
Contradiction!

Therefore, the membership problem
is undecidable.

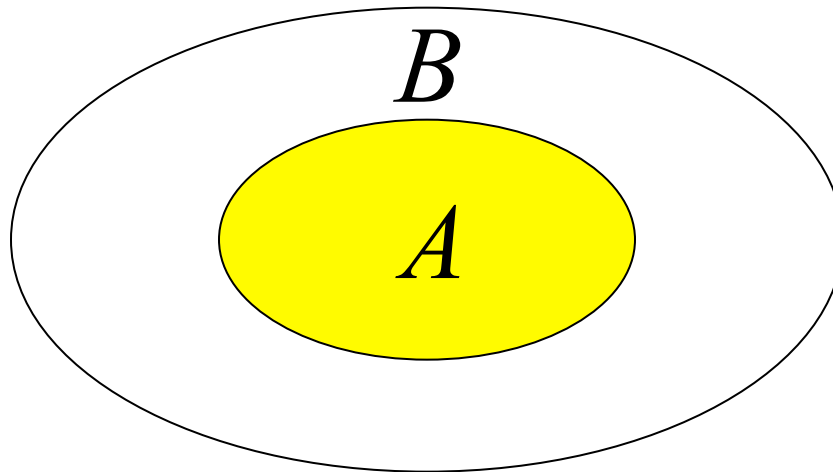
END OF PROOF

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

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.

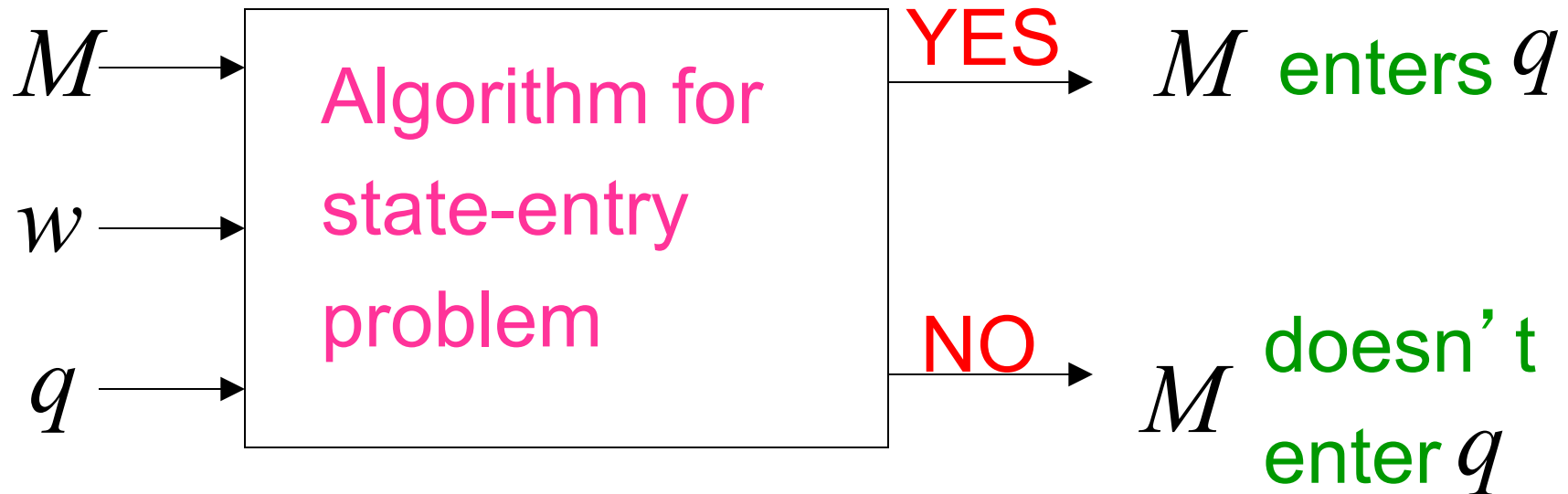
Proof

Reduce the halting problem to
the state-entry problem.

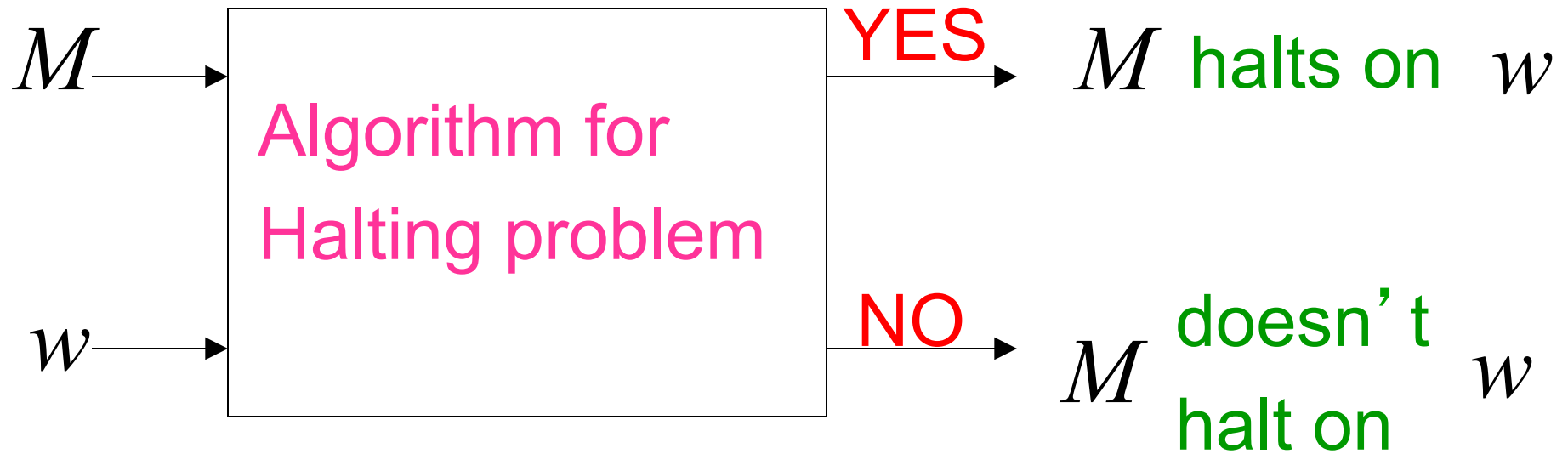
Suppose we have an algorithm
(Turing Machine)
that solves the state-entry problem.

We will construct an algorithm
that solves the halting problem.

Assume we have the state-entry algorithm:

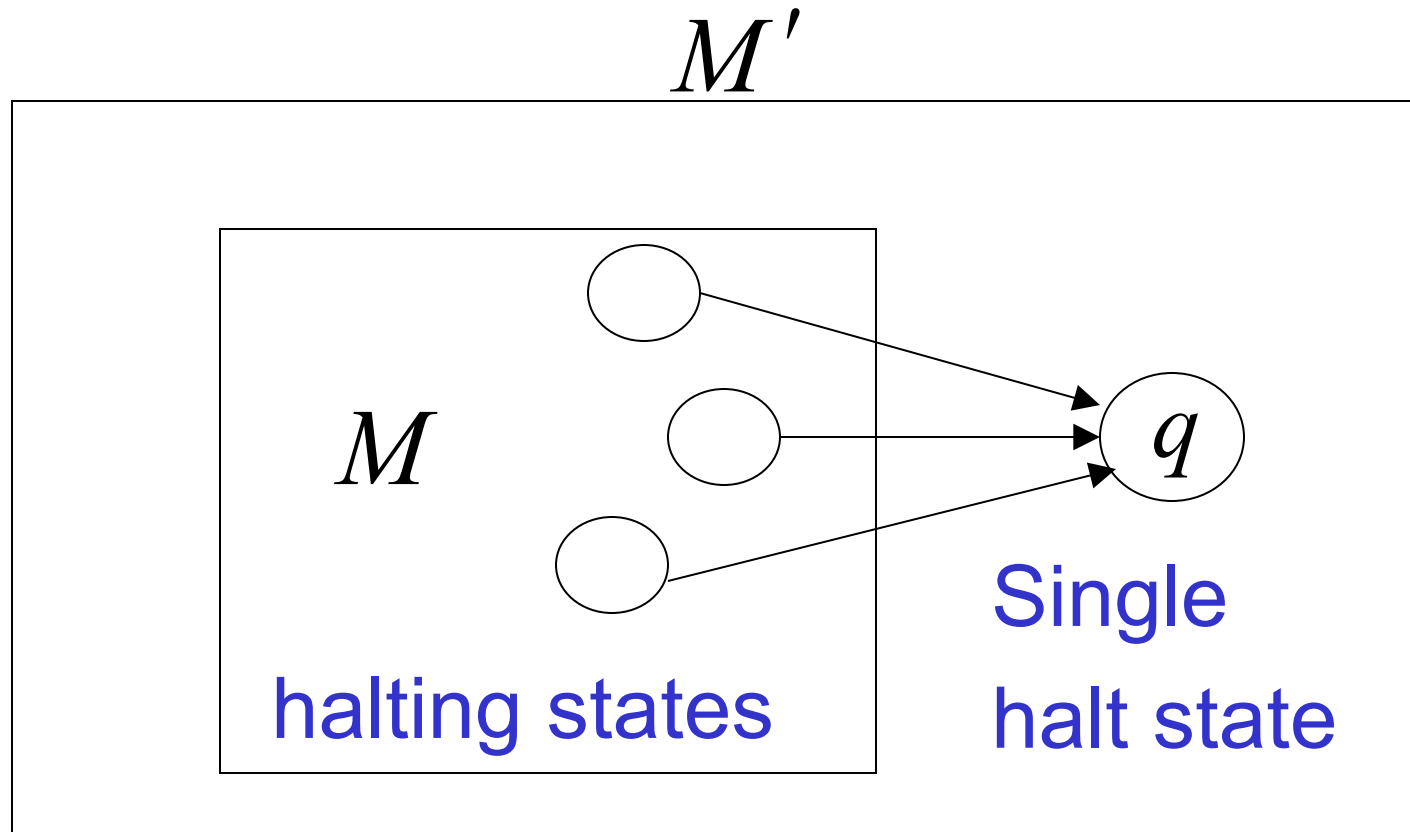


We want to design the halting algorithm:

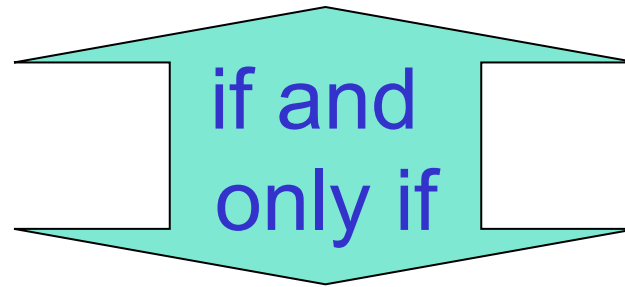


Modify input machine M

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



M halts



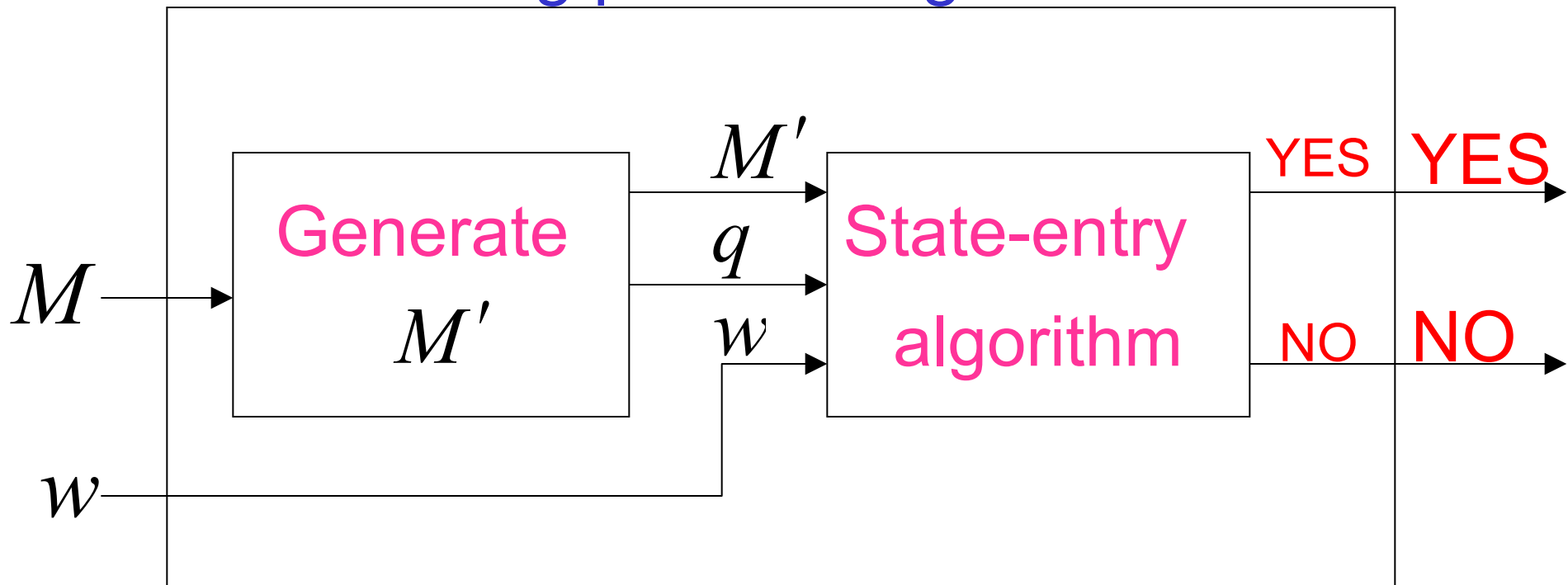
M' halts on state q

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

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.

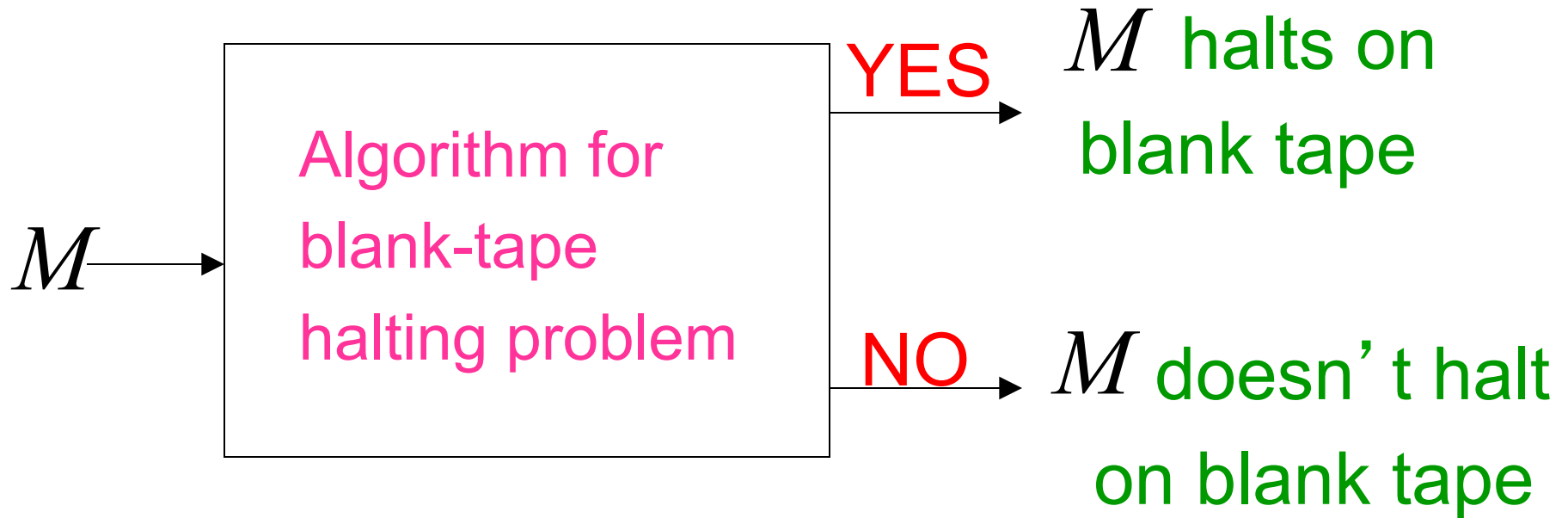
Proof

Reduce the halting problem to the blank-tape halting problem.

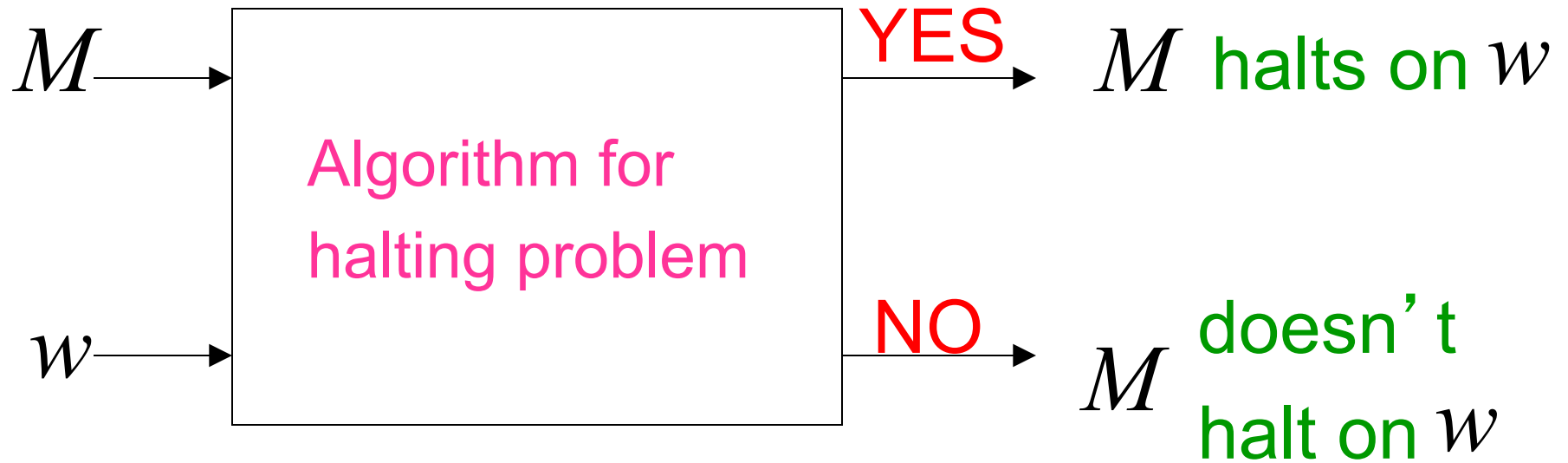
Suppose we have an algorithm
for the blank-tape halting problem.

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_w

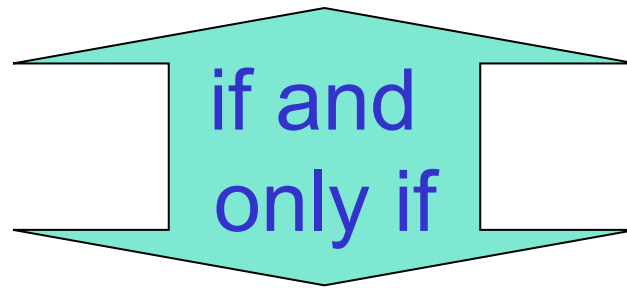
step 1

if blank tape
then write w

step2

execute M
with input w

M halts on input string w



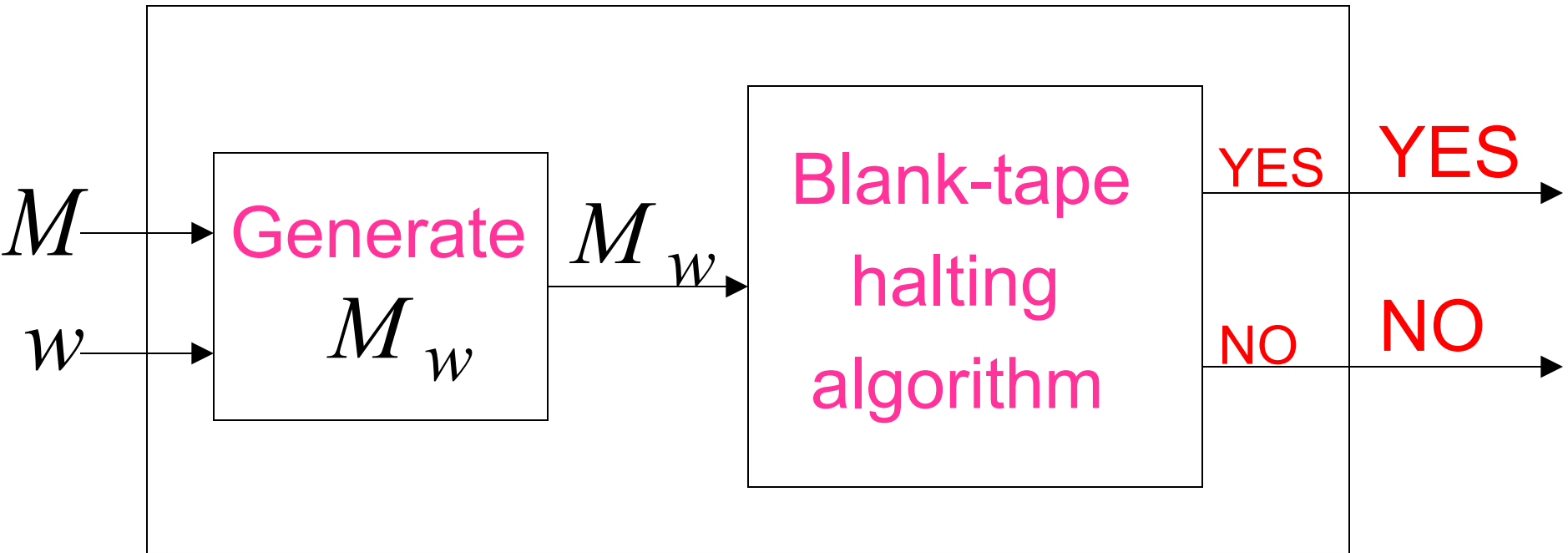
M_w halts when started with blank tape.

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

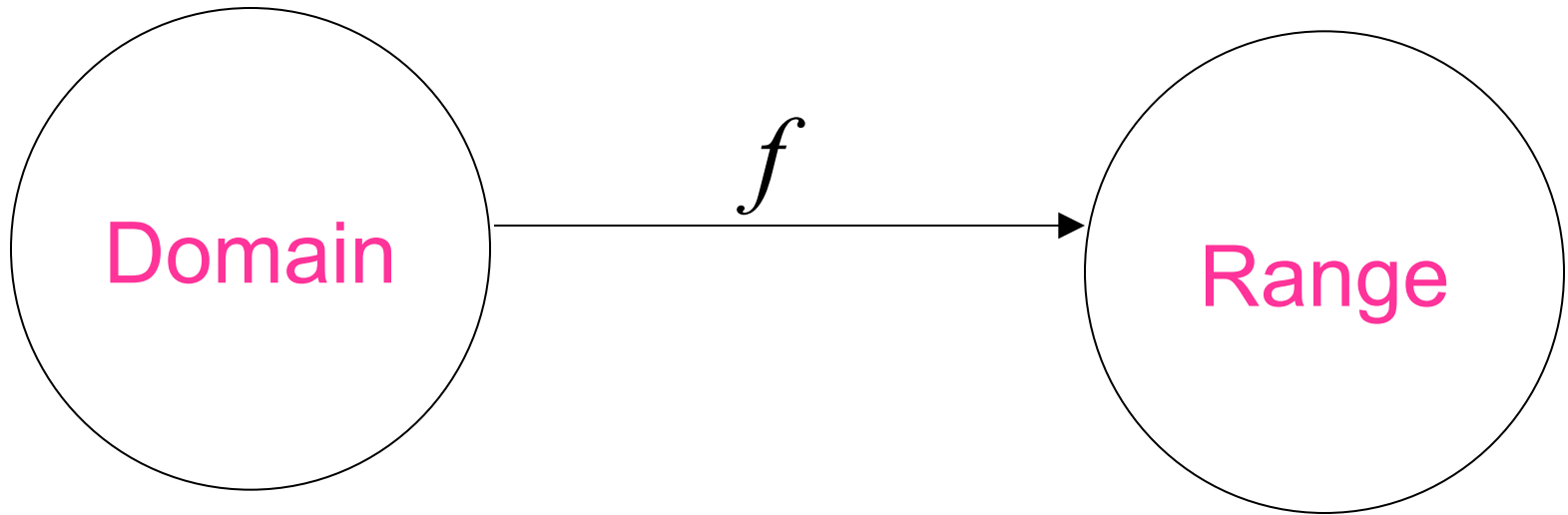
Does machine M halt when starting on blank tape?

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.

Example

An uncomputable function:

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

Theorem

Function $f(n)$ is uncomputable.

Proof

Assume to the contrary 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 must be decidable.

However, we know that the blank-tape halting problem is undecidable.

Contradiction!

Therefore, function $f(n)$ is uncomputable.

END OF PROOF

Rice' s Theorem

Definition

Non-trivial property 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.

Rice's Theorem

If Ω is a set of Turing-acceptable languages that contain some but not all such languages, no TM can decide for an arbitrary Turing-acceptable language L if L belongs to Ω or not.

Now we will prove some non-trivial properties
without using Rice's theorem.

Theorem

For any recursively enumerable language L it is **undecidable** whether it is empty.

Proof

We will reduce the membership problem to the problem of deciding whether L is empty.

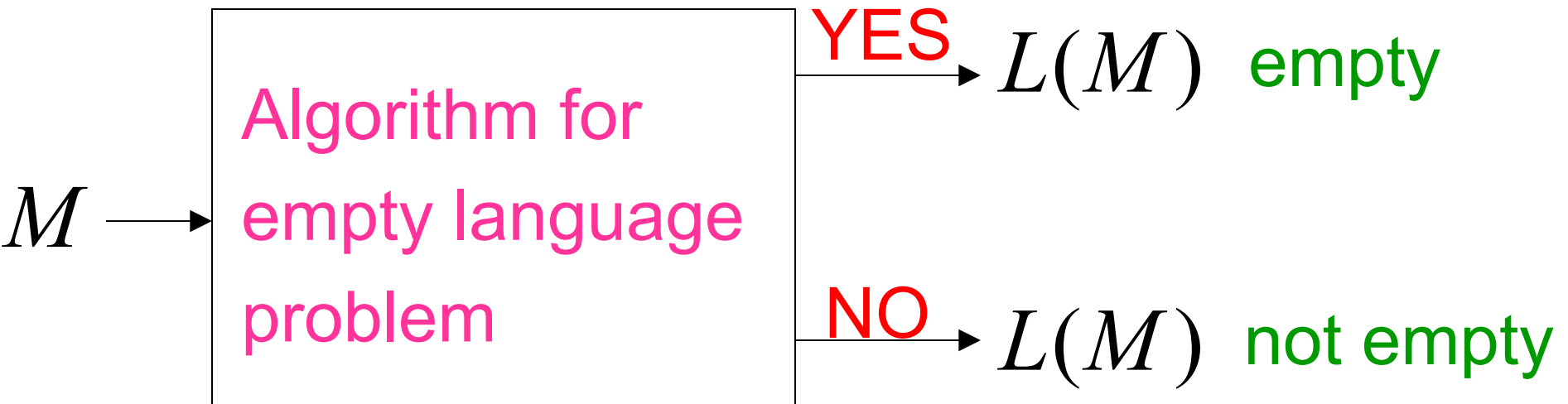
Membership problem:

Does machine M accept string w ?

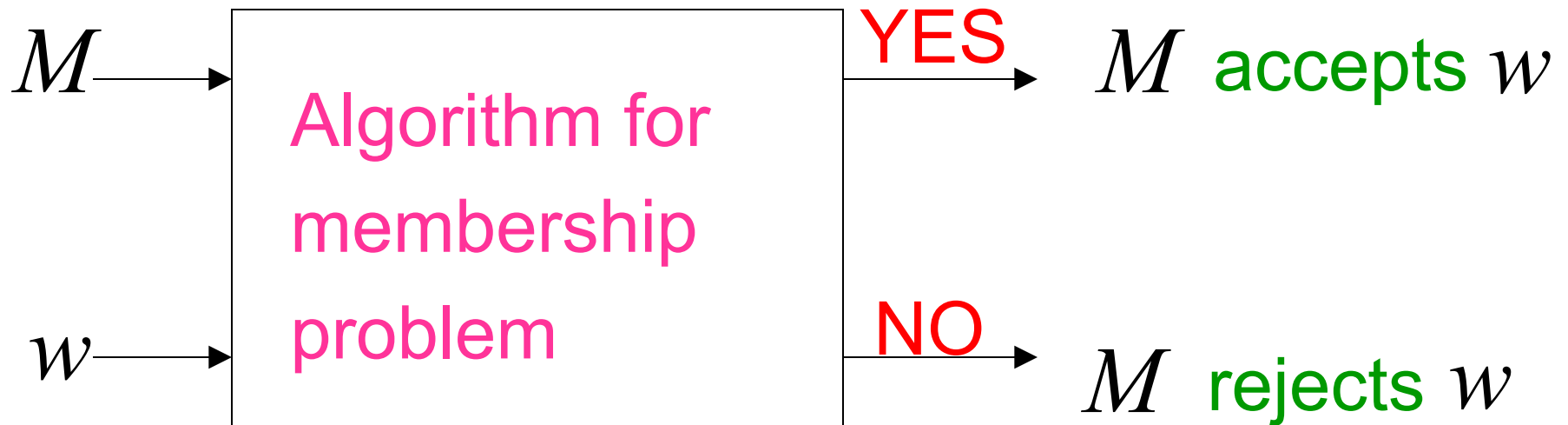
Let M be the machine that accepts L

$$L(M) = L$$

Assume we have the empty language algorithm:



We will design the membership algorithm:

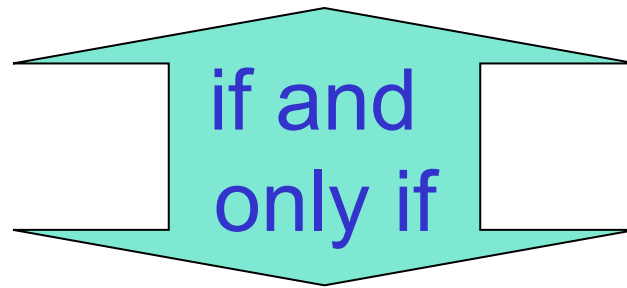


First construct machine M_w :

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

Accept if original input is
the same as w .

$$w \in L$$



$L(M_w)$ is not empty

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

Algorithm for membership problem

Inputs: machine M and string w

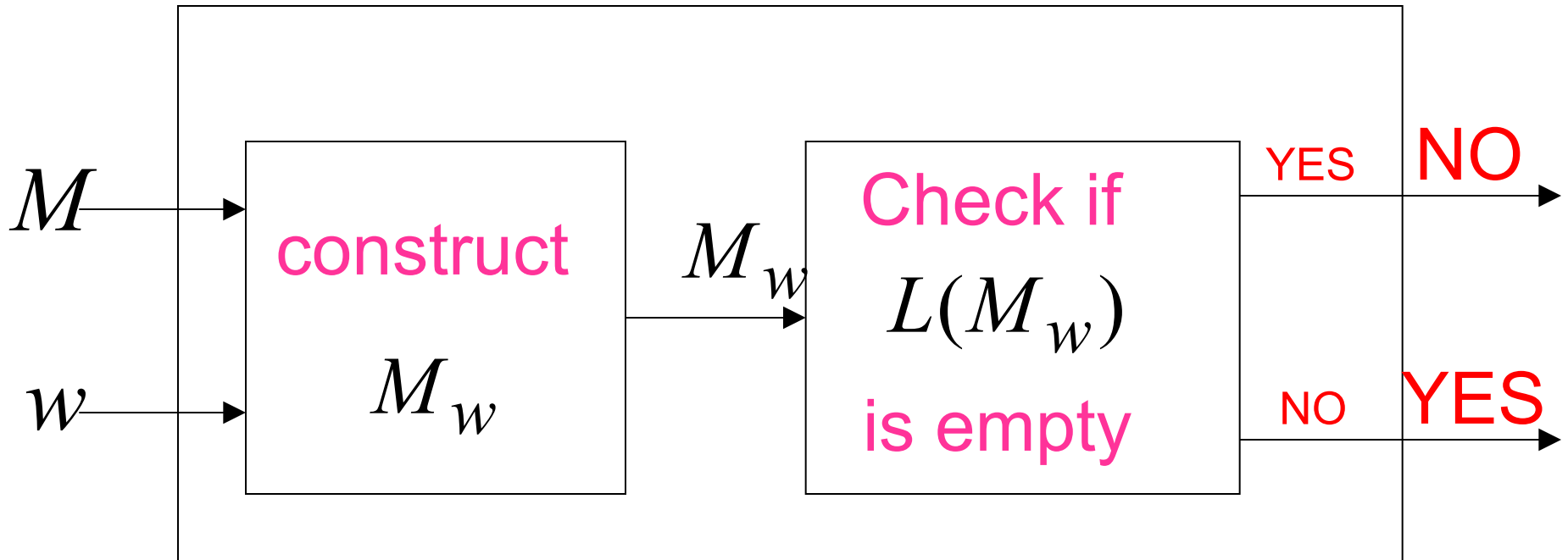
1. Construct M_w

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

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

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

Membership algorithm



We reduced the membership problem to the empty language problem.

Since the membership problem is undecidable, the empty language problem is also undecidable.

END OF PROOF

Theorem

For a recursively enumerable language L it is undecidable to determine whether L is finite.

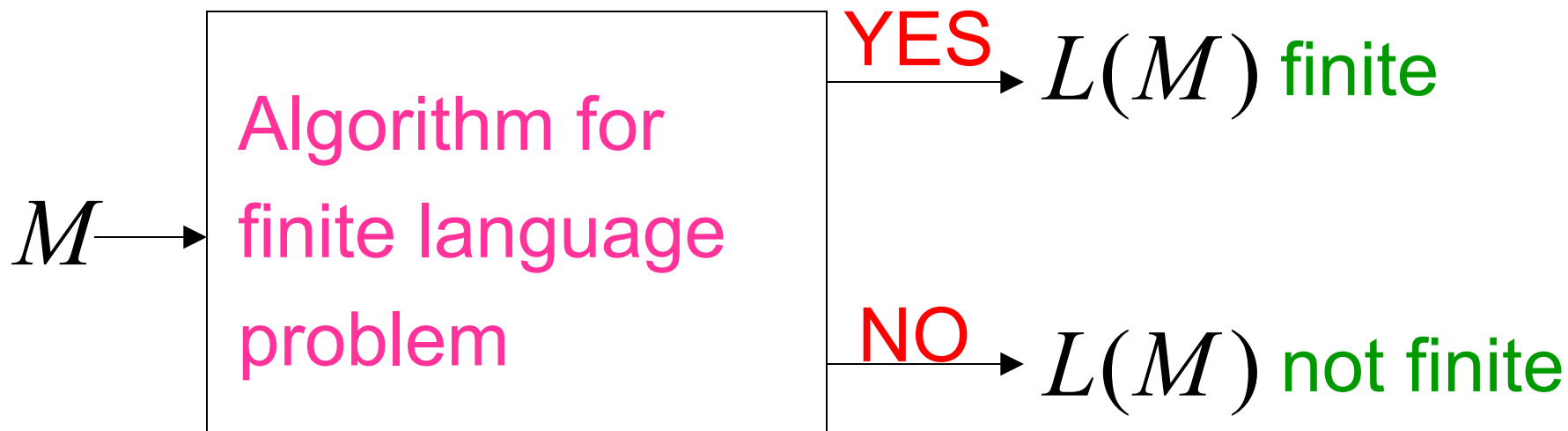
Proof

We will reduce the halting problem to the finite language problem.

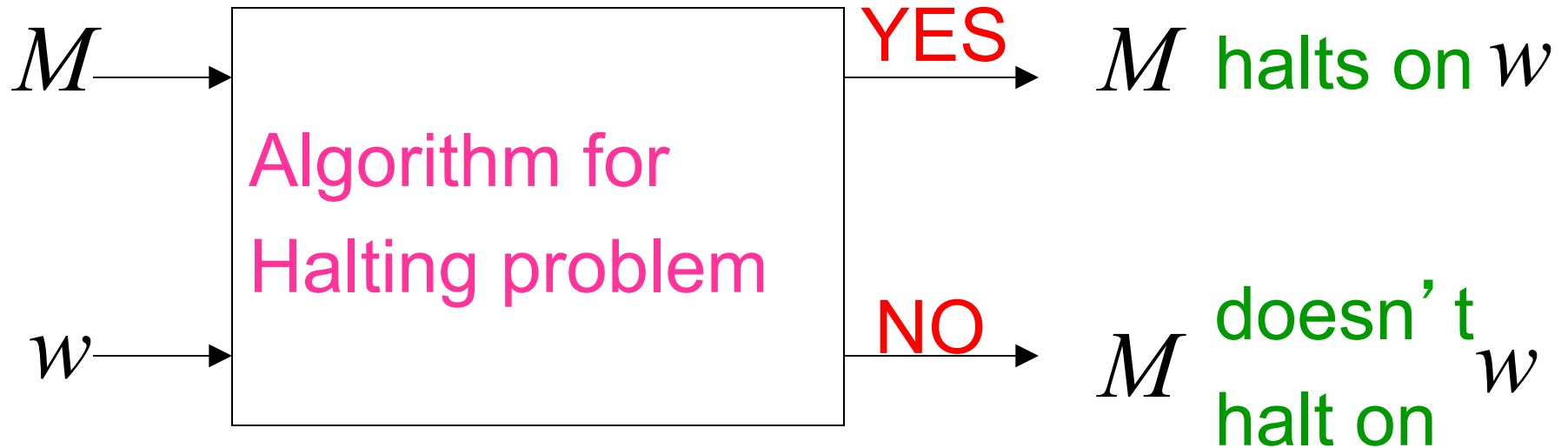
Let M be the machine that accepts L

$$L(M) = L$$

Assume we have the finite language algorithm:



We will design the halting problem algorithm:



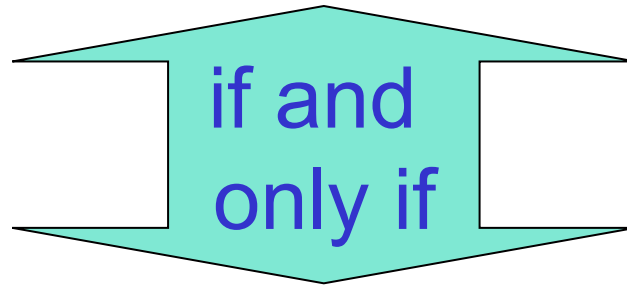
First construct machine M_w .

Initially, simulates M on input w .

When M enters a halt state,
accept any input (infinite language).

Otherwise accept nothing (finite language).

M halts on w



$L(M_w)$ is not finite.

Algorithm for halting problem:

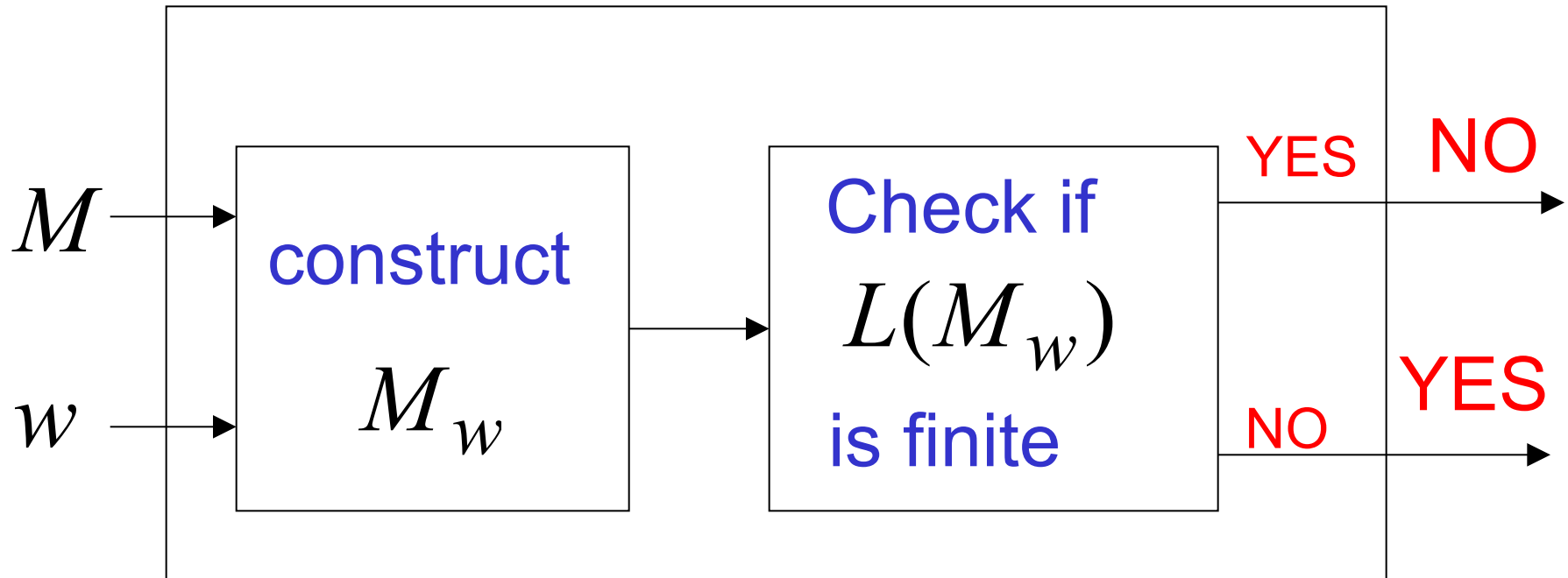
Inputs: machine M and string w

1. Construct M_w
2. Determine if $L(M_w)$ is finite

YES: then M doesn't halt on w

NO: then M halts on w

Machine for halting problem



We reduced the halting problem
to the finite language problem.

Since the halting problem is undecidable,
the finite language problem is
also undecidable.

END OF PROOF

Theorem

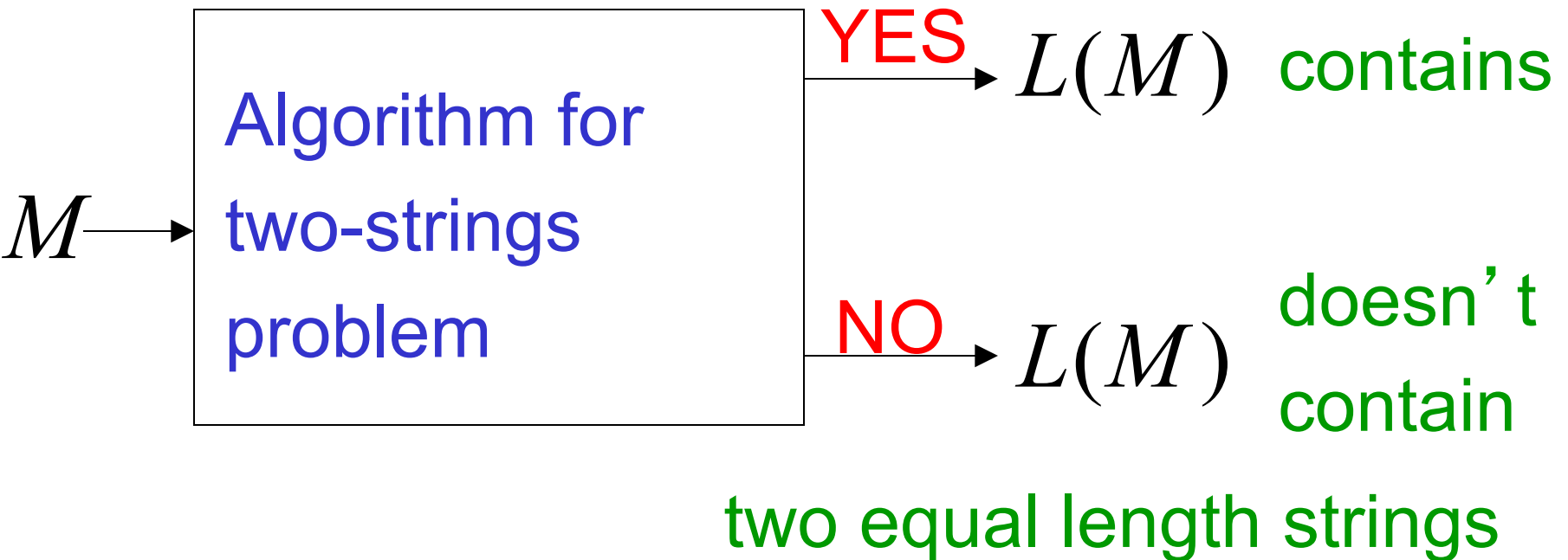
For a recursively enumerable language L
it is undecidable whether L contains
two different strings of same length.

Proof

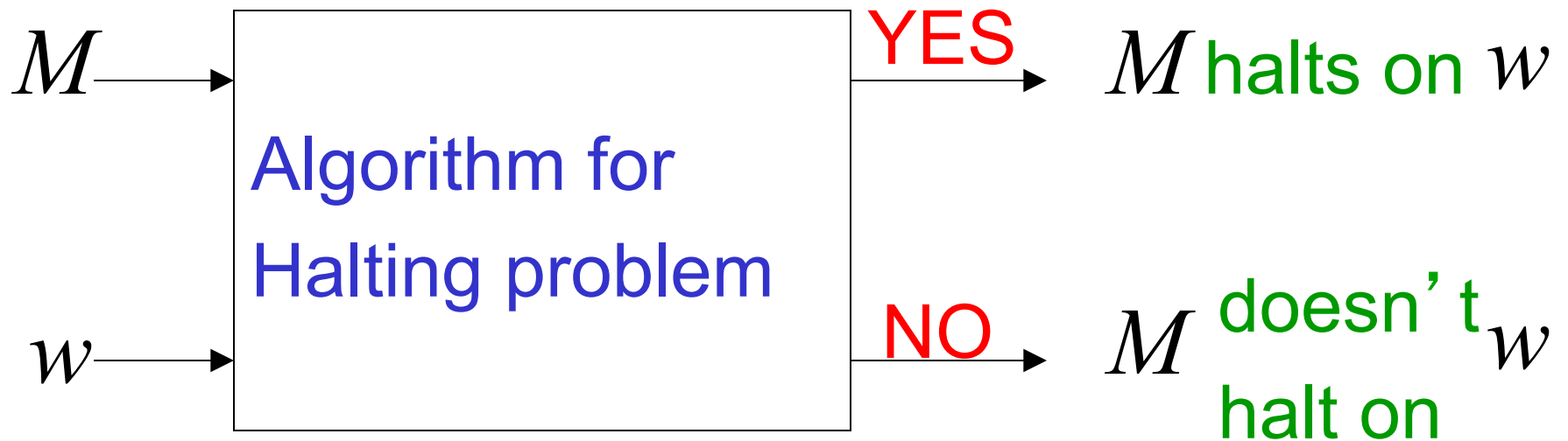
We will reduce the halting problem
to the two strings of equal length- problem.

Let M be the machine that accepts L
 $L(M) = L$

Assume we have the two-strings algorithm:



We will design the halting problem algorithm:



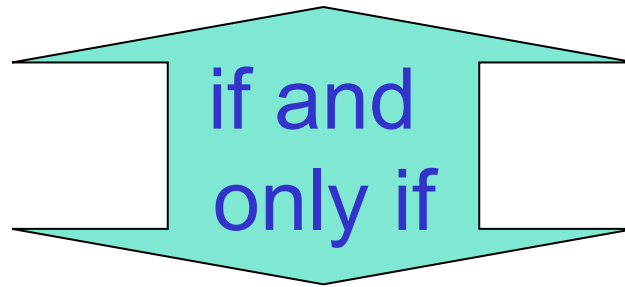
First construct machine M_w .

Initially, simulates M on input w .

When M enters a halt state,
accept symbols a or b .

(two equal length strings)

M halts on w



M_w accepts a and b

(two equal length strings)

Algorithm for halting problem

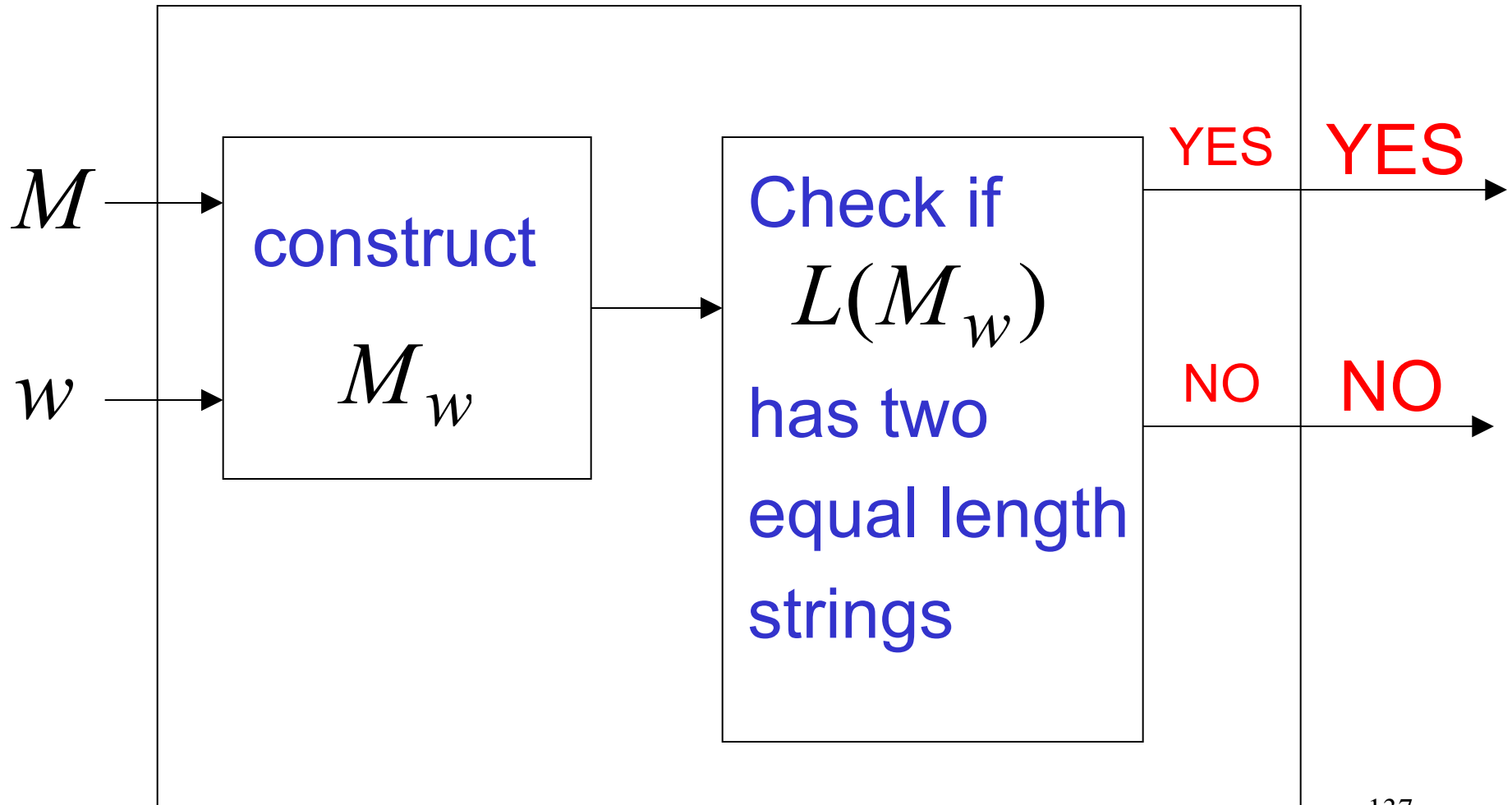
Inputs: machine M and string w

1. Construct M_w
2. Determine if M_w accepts two strings of equal length

YES: then M halts on w

NO: then M doesn't halt on w

Machine for halting problem



We reduced the halting problem
to the two strings of equal length – problem.

Since the halting problem is undecidable,
the two strings of equal length problem is
also undecidable.

END OF PROOF