class20.pdf

- Page 2. The Chomsky Hierarchy. Linz 6^{th}, §11.2 Unrestricted Grammars. Proof of equivalence in textbook.
- Page 8. CSL and LBAs. Linz 6^{th}, §11.3
- Page 15. Decidability. Linz 6^{th}, Thm 12.1. Proof by contradiction that the halting problem is undecidable.
  1. By constructing an impossible machine
  2. By proving no such thing as a recursive language

The Chomsky Hierarchy

Unrestricted Grammars:

Productions

$$u \rightarrow v$$

String of variables and terminals

String of variables and terminals

Example unrestricted grammar:

$$S \rightarrow aBc$$

$$aB \rightarrow cA$$

$$Ac \rightarrow d$$

Theorem:

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

Theorem 11.6, Linz, 6th, page 293. Any language generated by an unrestricted grammar is recursively enumerable.

Theorem 11.7, Linz, 6th, page 297. Any recursively enumerable language is generated by an unrestricted grammar.

It is no surprise the unrestricted grammars are recursive enumerable.

A grammar generates strings by a well-defined algorithmic process, so the derivations can be done on a Turing machine.

(An appeal to Church's thesis.)

To show the converse, we describe how any Turing machine can be mimicked by an unrestricted grammar.

The idea is relatively simple, but the complete construction is arduous. Simulate the instantaneous descriptions.

If q0 w |-* x qf w , then q0 w =>* x qf y

Cleverly keep a copy of w so that the productions can generate *after* verifying the TM's transitions to a final state.

Context-Sensitive Grammars

Linz 6th, Section 11.3 Context-Sensitive Grammars and Languages

Definition 11.4, page 300. [See next slide]

Definition 11.5, page 300. *A* language L is said to be context sensitive if there exists a context-sensitive grammar G st L (minus epsilon) equals L(G).

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:

$$S \to abc \mid aAbc$$

$$Ab \to bA$$

$$Ac \to Bbcc$$

$$bB \to Bb$$

$$aB \to aa \mid aaA$$

Derivation of $a^3b^3c^3$

$$S \to abc \mid aAbc$$
$$Ab \to bA$$
$$Ac \to Bbcc$$
$$bB \to Bb$$
$$aB \to aa \mid aaA$$

S => aAbc => abAc

B travels left to create the corresponding a

=> abBbcc => aBbbcc

A travels right to the first c

$\Rightarrow$ aaAbbcc =>* aabbAcc

B travels left to create the corresponding a

=> aabbBbccc => aabBbbccc => aaBbbbccc =>

finish with aB -> aa

=> aaabbbccc

Theorem:

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

Linz 6th, §11.3 Thms 11.8 and 11.9, page 302.
The proofs are essentially the same as for TMs and unrestricted grammars.

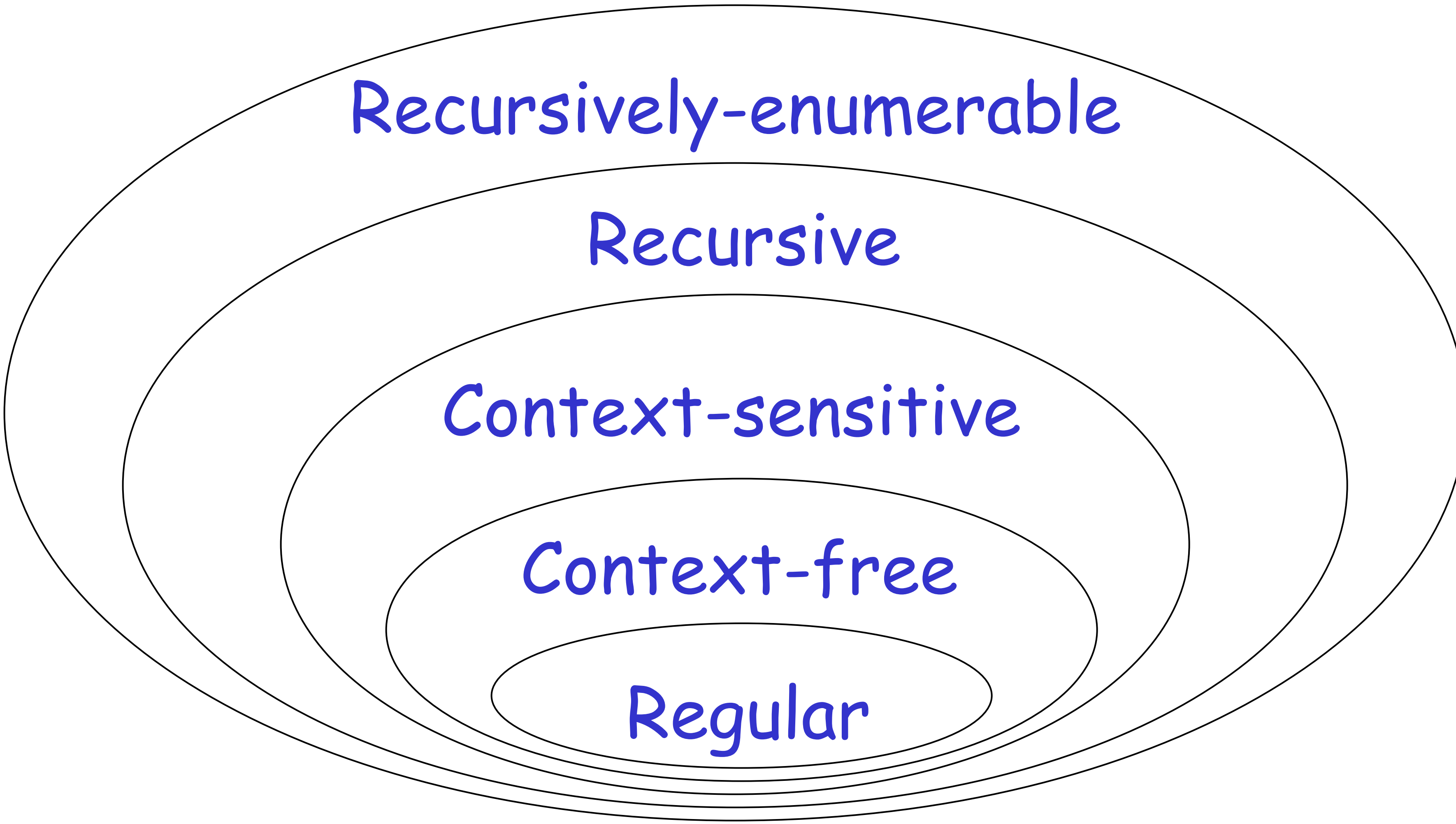
Observation:

Linz 6th, Theorem 11.11, page 303. There is a recursive language that is not context sensitive.

Encode CSG and diagonalize. Membership for L={w_i st CSG G_i and w_i not in L(G_i)} is decidable.

The Chomsky Hierarchy

Non-recursively enumerable



Decidability

Linz 6th, Chapter 12: Limits of Algorithmic Computation, page 309ff

A property P of strings is said to be *decidable* if the set of all strings having property P is a recursive set; that is, if there is a total Turing machine that accepts input strings that have property P and rejects those that do not.

$$P \text{ is decidable} \iff \{x \mid P(x)\} \text{ is recursive.}$$

$$A \text{ is recursive} \iff \text{“}x \in A\text{” is decidable,}$$

Kozen

Consider problems with answer YES or NO

Examples:

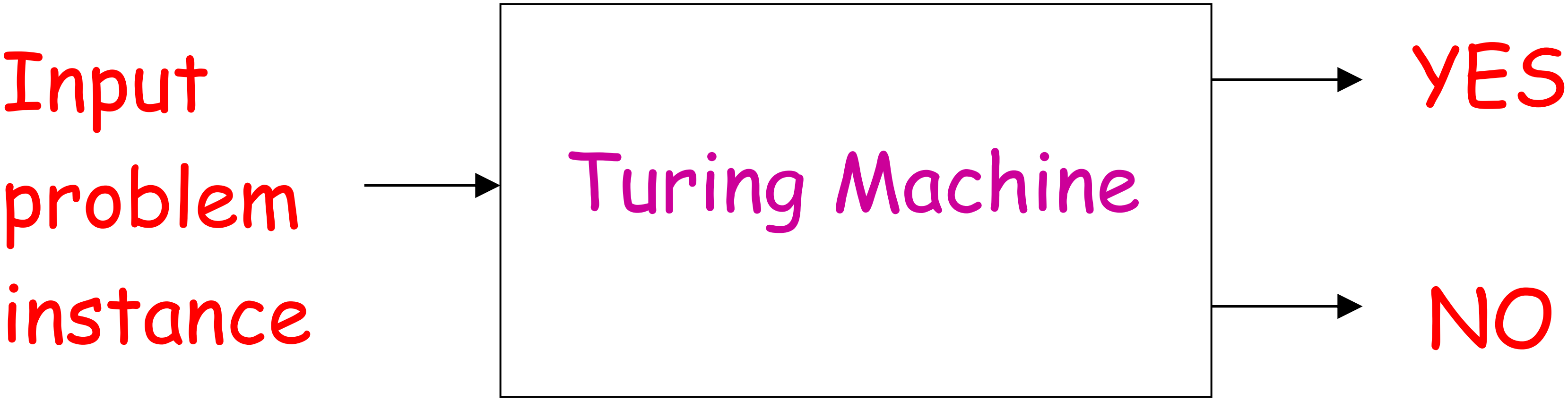
- Does Machine M have three states ?
- Is string w a binary number?
- Does DFA M accept any input?

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



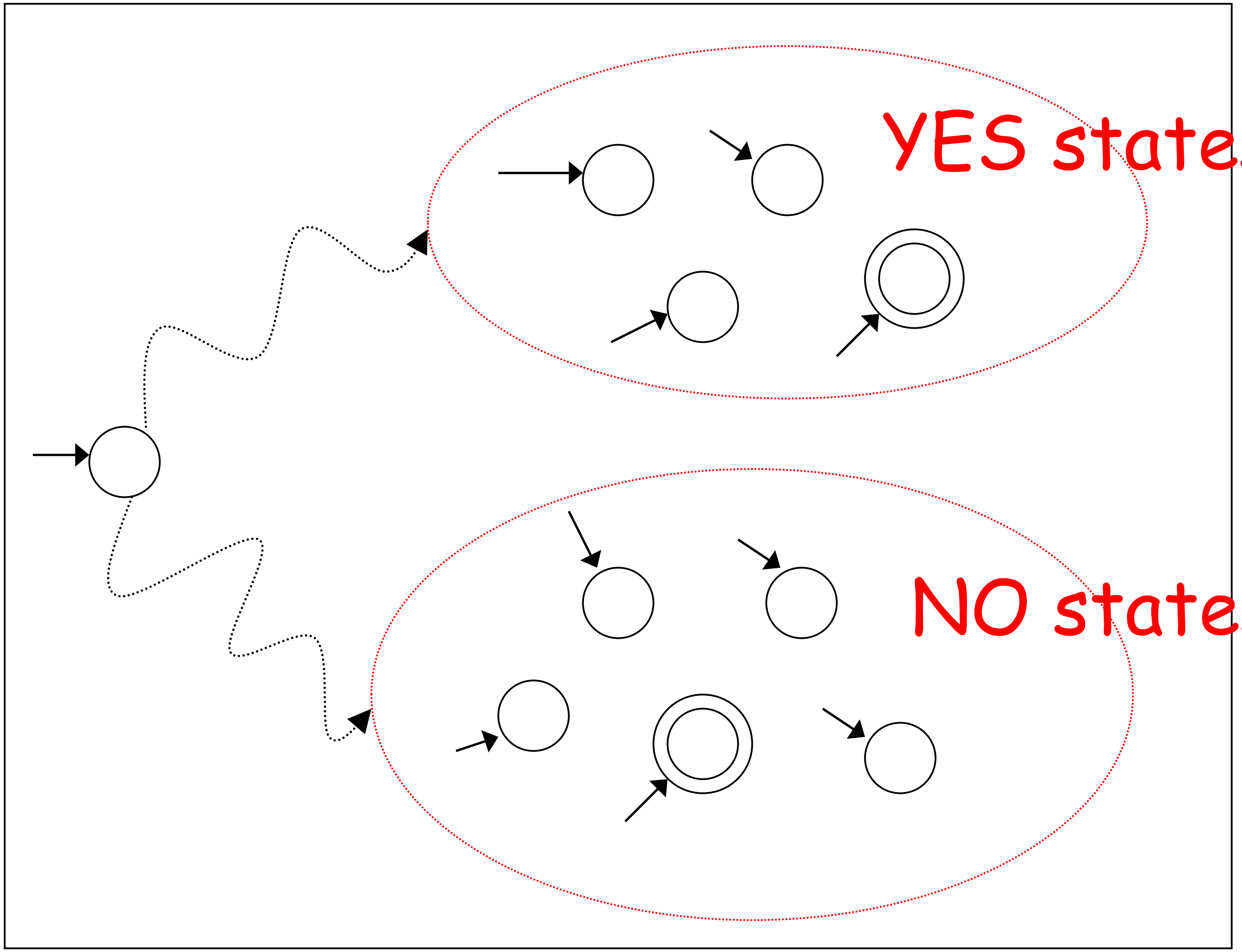
The machine that decides a problem:

- If the answer is YES
  then halts in a <u>yes state</u>

- If the answer is NO
  then halts in a <u>no state</u>

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

[What does the author mean?]

No harm in assuming YES states are final.

Decidable = Recursive

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, and

•String w

Question: Does M accept w ?

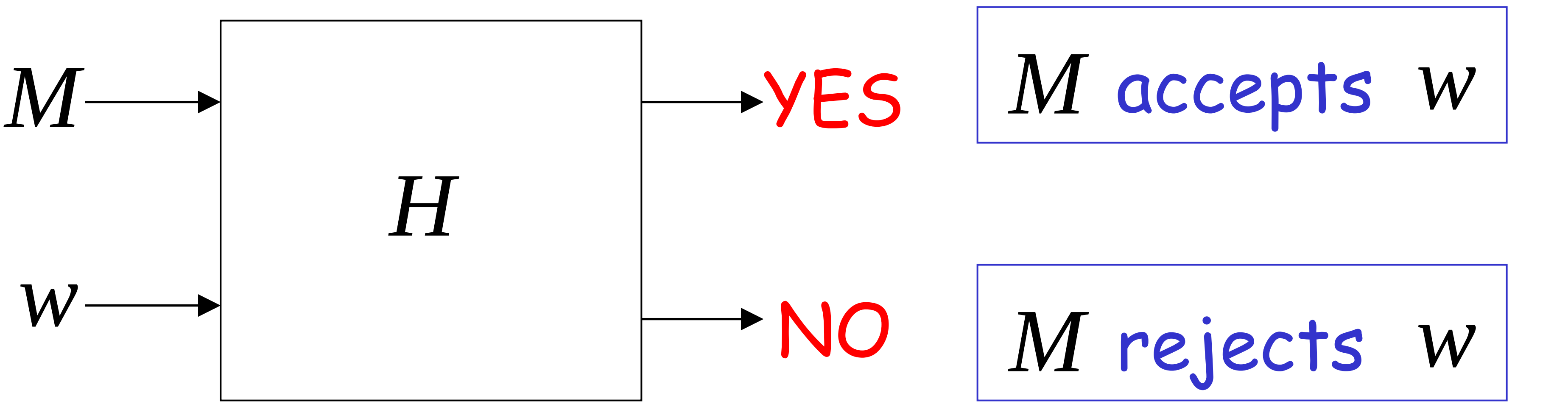
$$w \in L(M)?$$

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

Assume there exists a Turing Machine H that decides/solves the membership problem



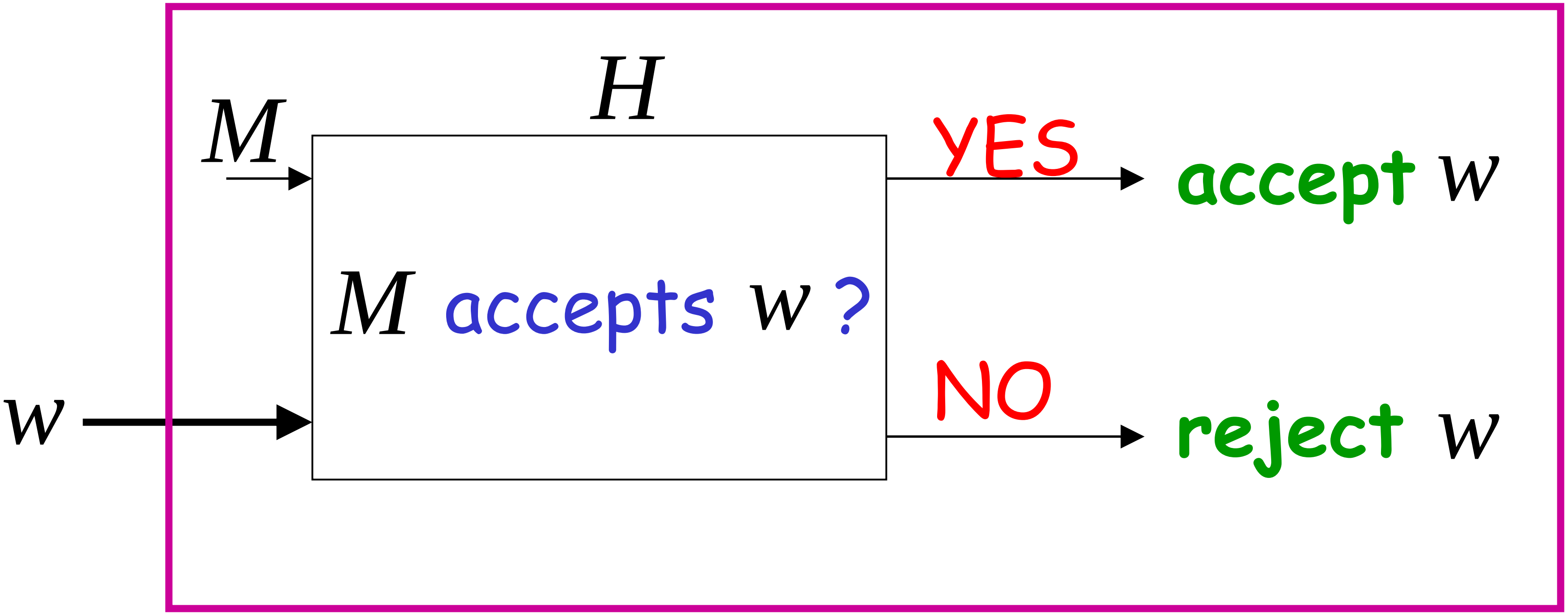
Let L be a recursively enumerable language

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

Another famous undecidable problem:

The halting problem

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*,

The Halting Problem

Input: • Turing Machine M, and

• String w

Question: Does M halt on input w?

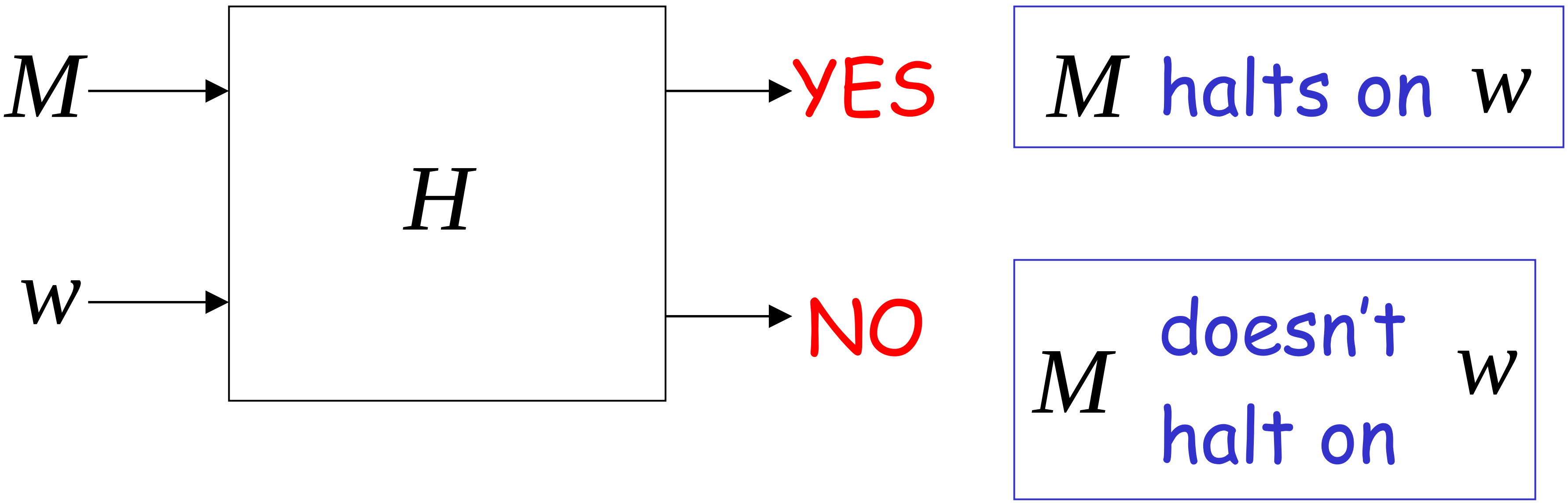
Theorem:

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

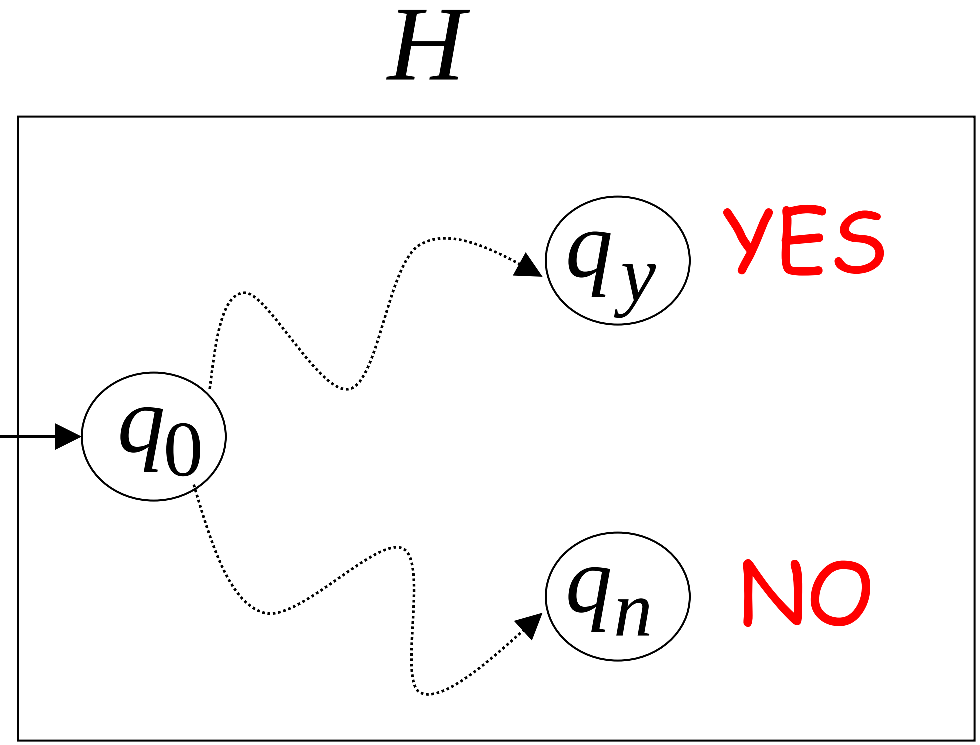
Thus, there exists Turing Machine H that solves the halting problem



Construction of H

Input:
initial tape contents

$w_M \quad w$



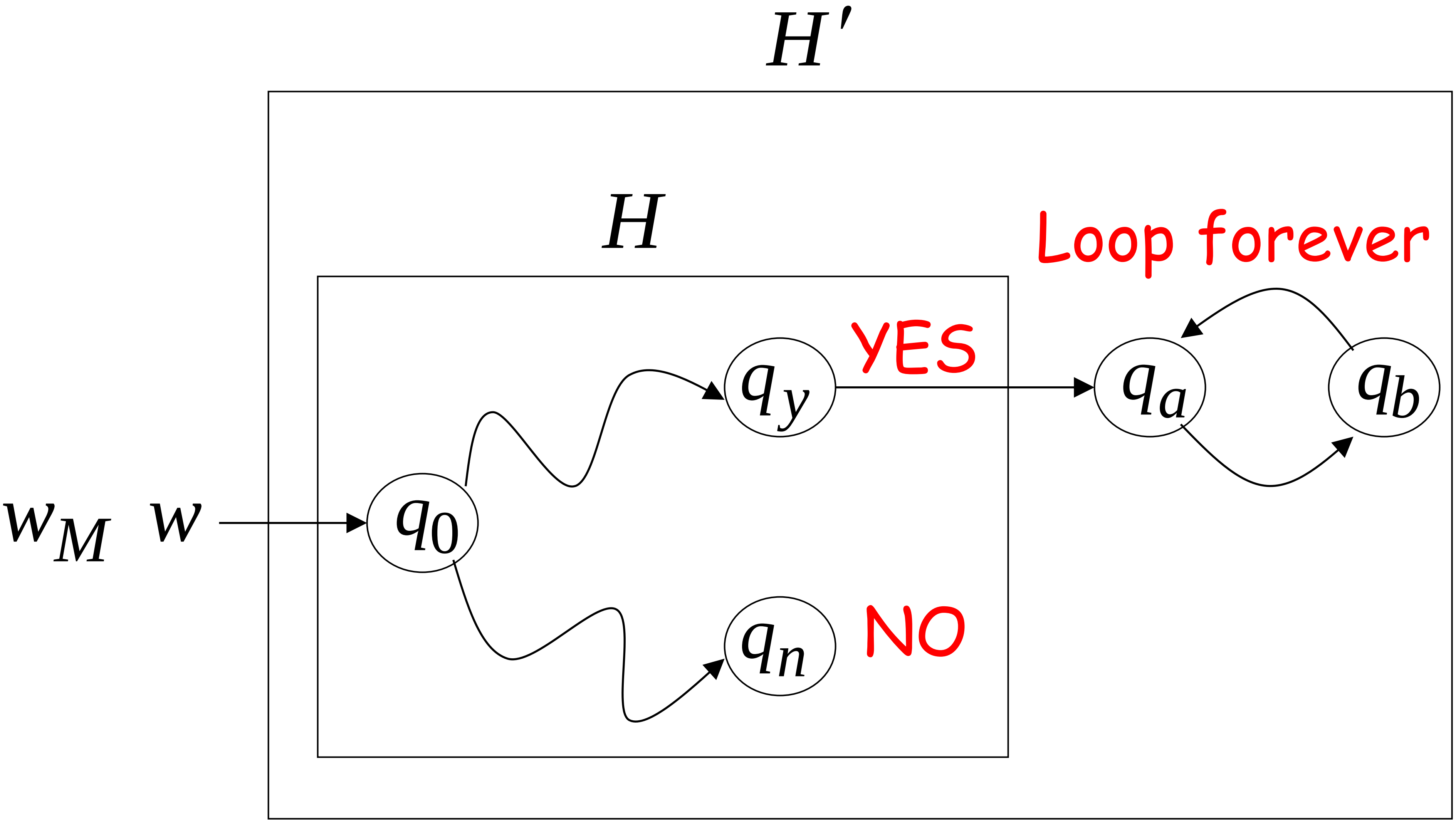
Encoding of M

String w

Construct machine H' :

If H returns YES then loop forever

If H returns NO then halt


H'
H
Loop forever
YES
NO
q_y
q_a
q_b
q_0
q_n
w_M w

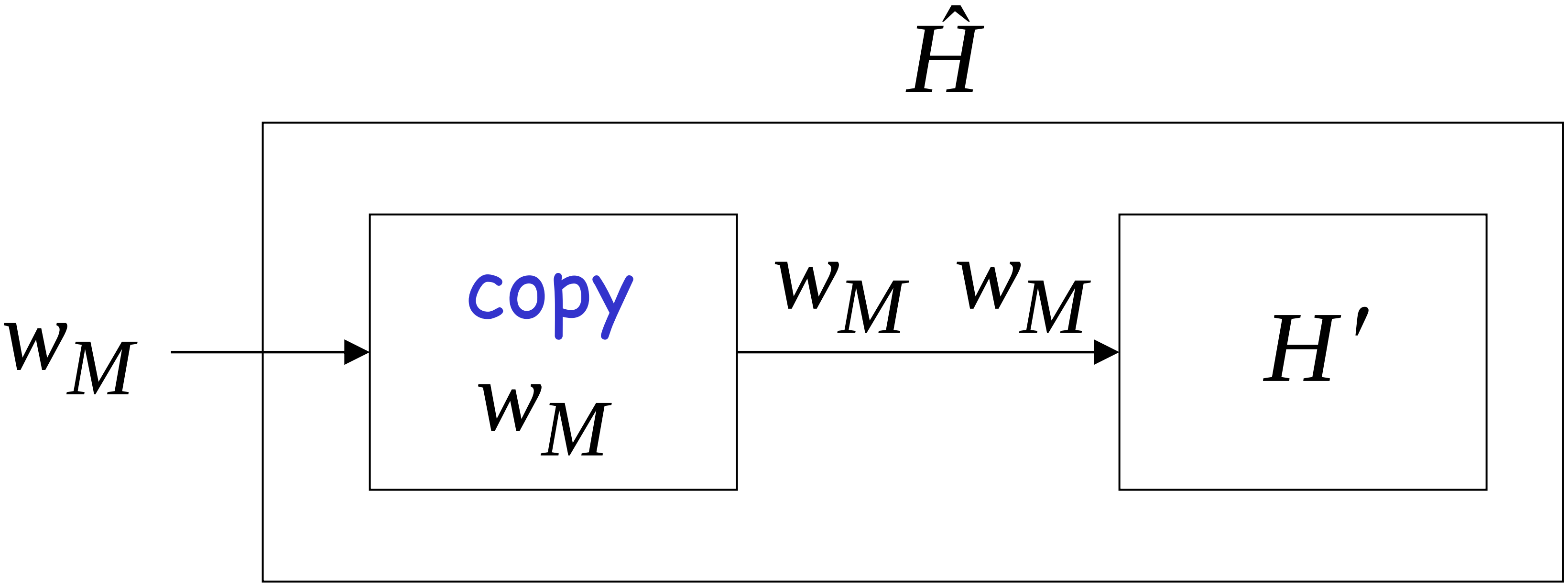
Construct machine $\hat{H}$:

Input: w_M (machine M)

If M halts on input w_M

Then loop forever

Else halt


$\hat{H}$
w_M
copy
w_M
w_M w_M
H'

Run machine $\hat{H}$ with input itself:

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

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

<u>Then</u> loop forever

<u>Else</u> halt

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

If $\hat{H}$ halts then loops forever

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

NONSENSE !!!!!

Therefore, we have contradiction

The halting problem is undecidable

END OF PROOF #1

Another proof of the same theorem:

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

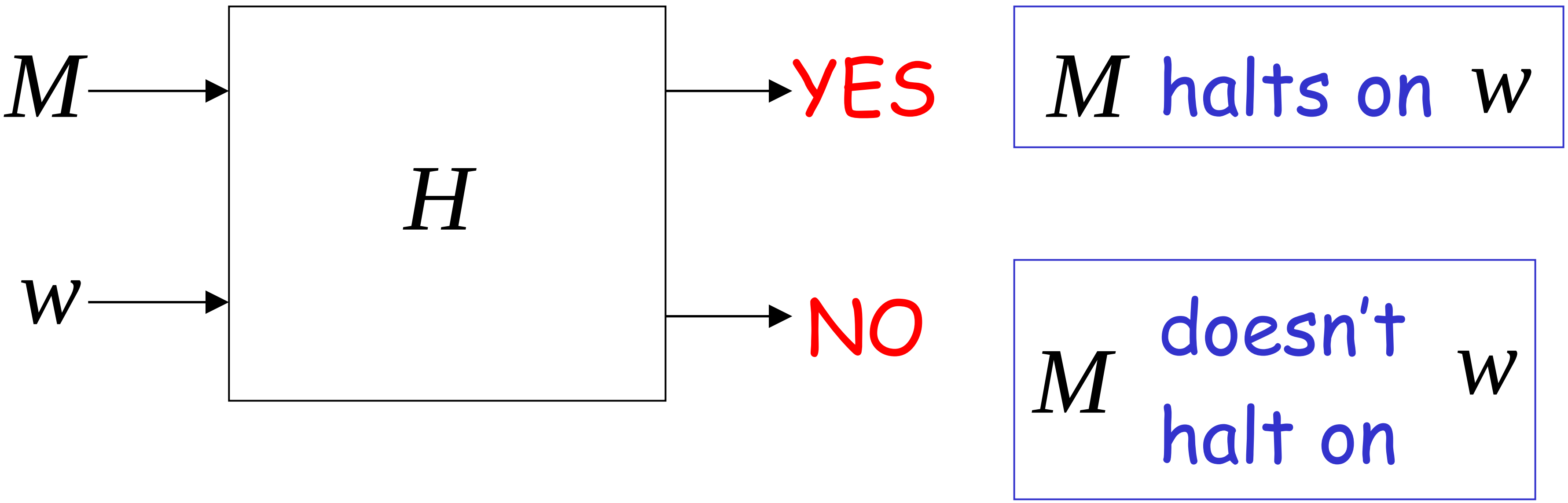
If $H=\{w\#enc(M) | w \in L(M)\}$ were recursive, then every r.e. set is recursive.

Theorem:

The halting problem is undecidable

Proof: Assume for contradiction that the halting problem is decidable

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

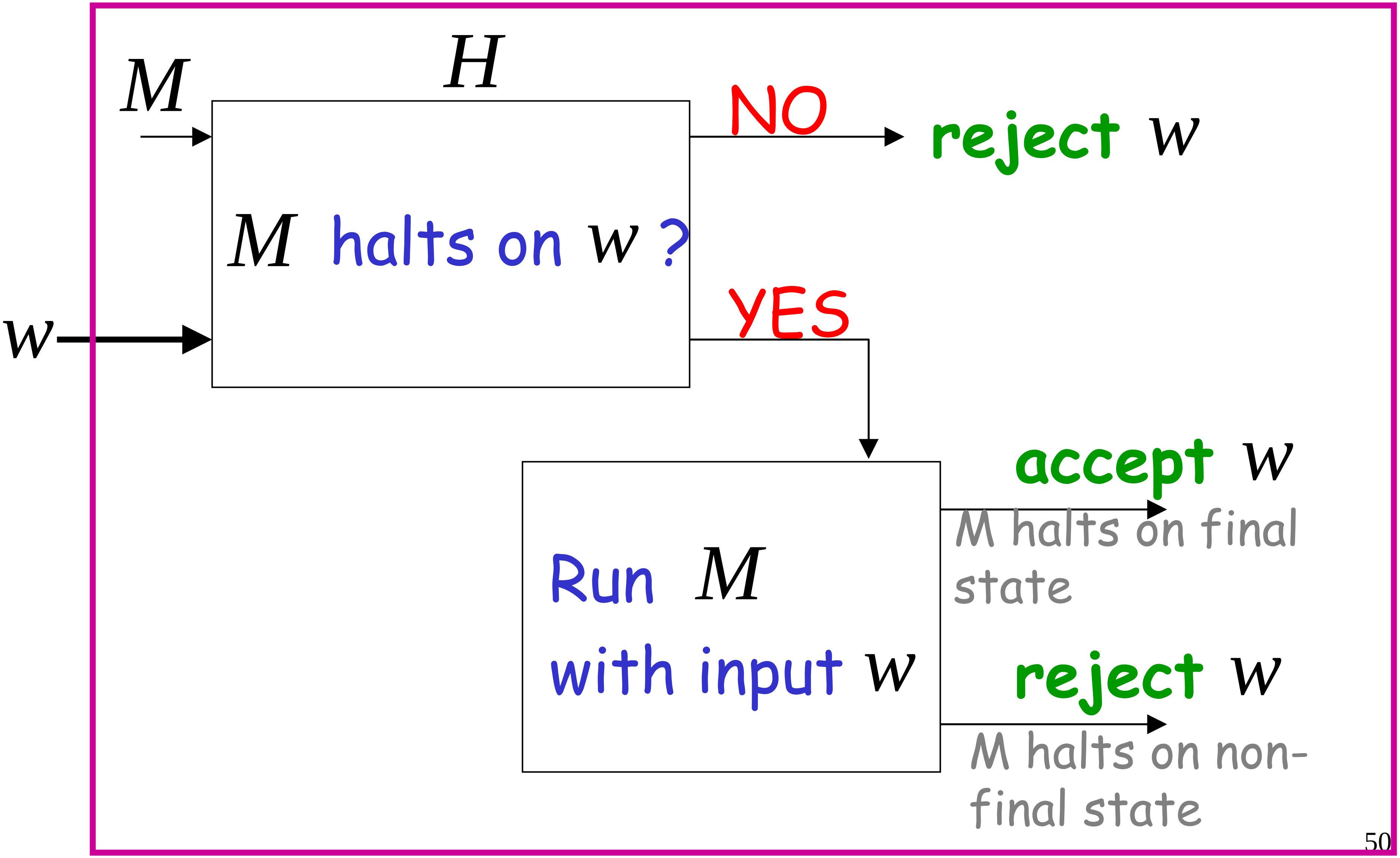


Let L be a recursively enumerable language

Let M be the Turing Machine that accepts L

we will describe a Turing machine that accepts L and halts on any input, proving that L is also recursive.

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 #2