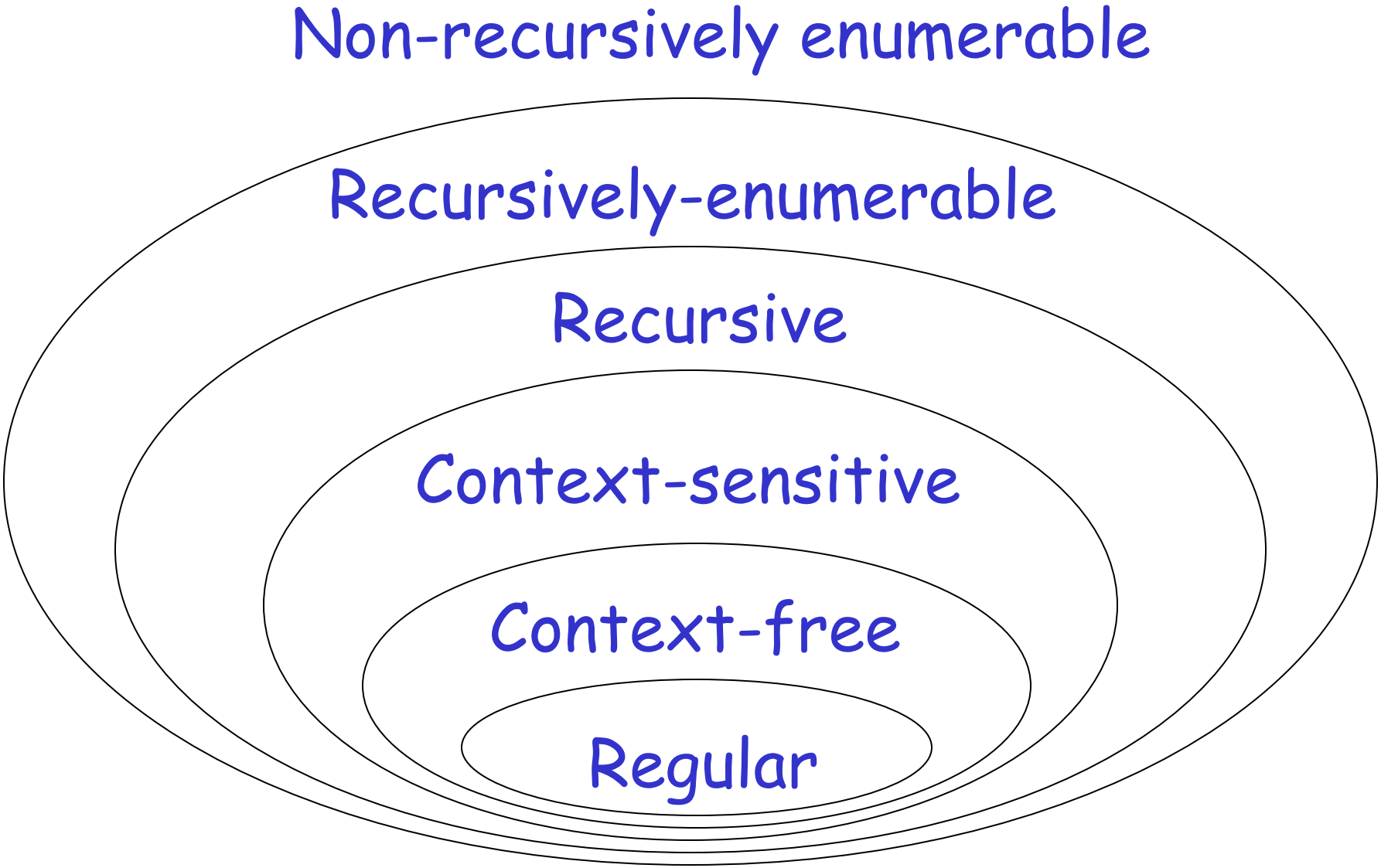
The Chomsky Hierarchy



Decidability

Consider problems with answer YES or NO

Examples:

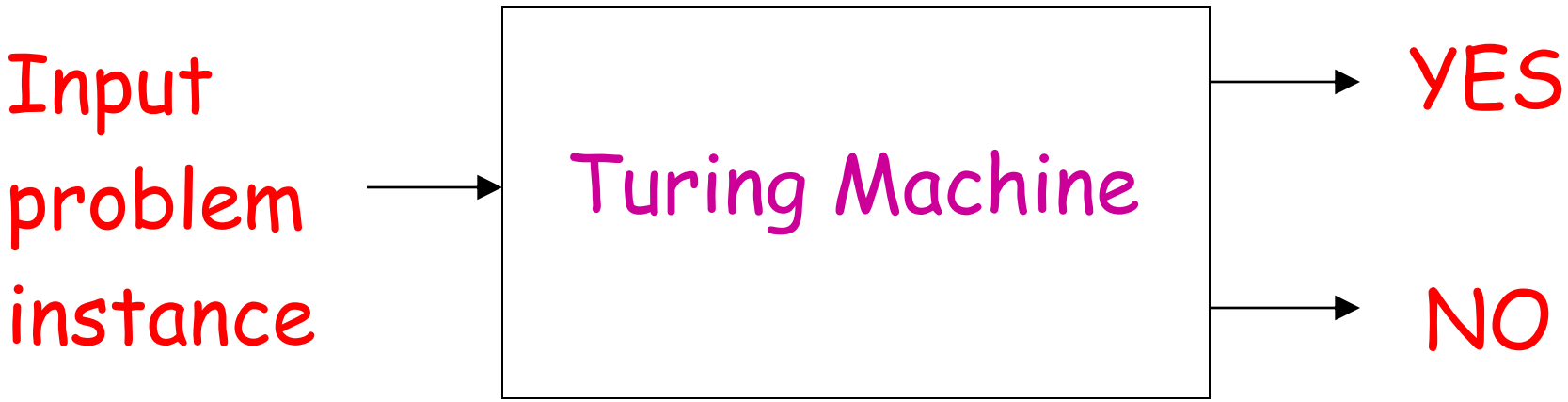
- Does Machine M have three states ?
- Is string w a binary number?
- Does DFA M accept any input?

A problem is decidable if some Turing machine decides (solves) 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 decides (solves) a problem answers YES or NO for each instance of the problem



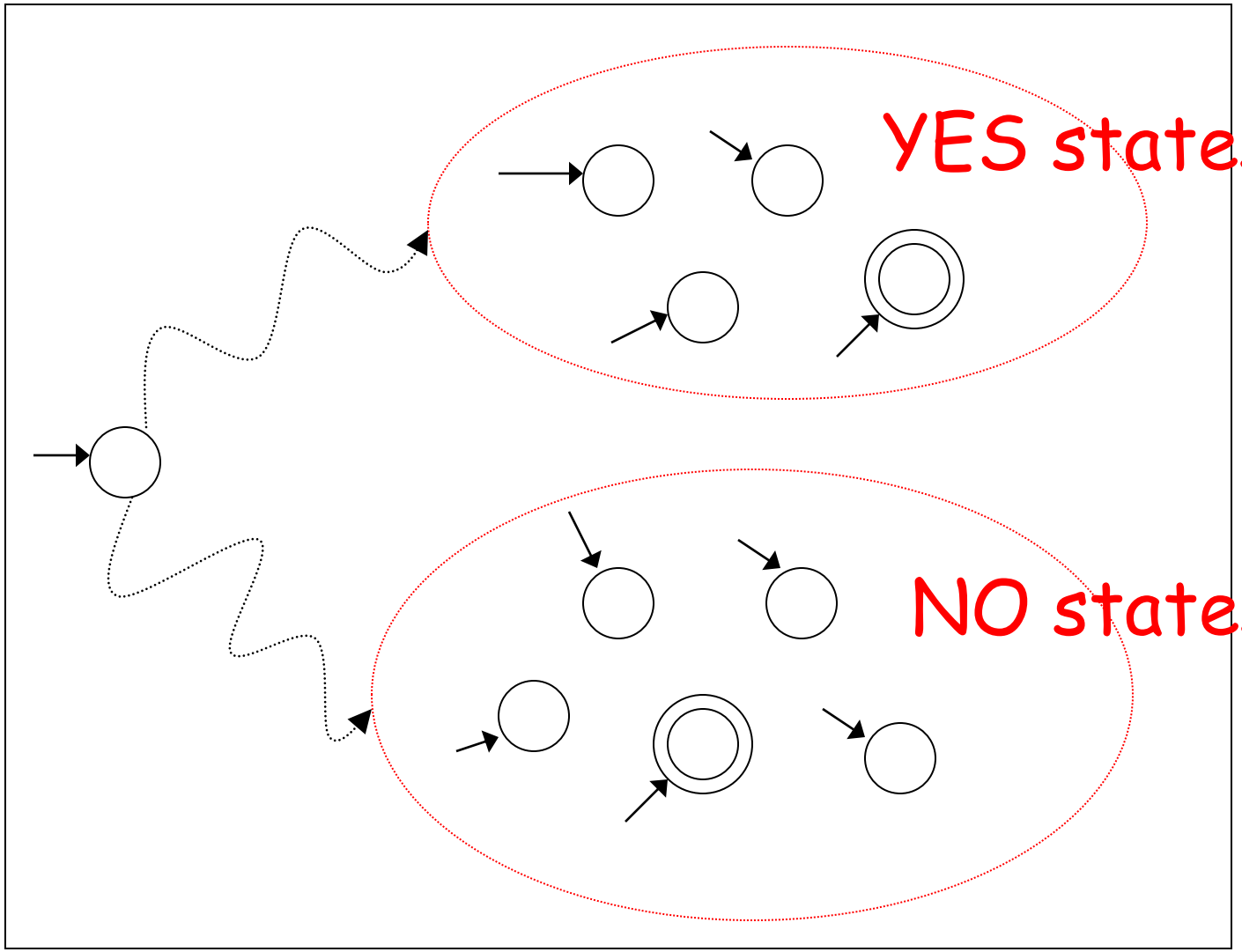
The machine that decides (solves) a problem:

- 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

Problems = Languages

Given any TM M and any input w, will M halts on w?

This is a problem.

Define

This is a language.

$$H = \{< M, w > | \; M \quad halts \quad on \quad w\}$$

The fact

The problem and the language are equivalent!

Problems = Languages

Given any string w, will w have property P?

This is a problem.

Define

This is a language.

$$H = \{w \mid w \quad has \quad property \quad P\}$$

Conclusion:

A problem is a language.

A language is a problem.

A decidable problem

A_{DFA}={<B,w>|B is a DFA that accepts input string w}

Theorem 4.1 A_{DFA} is a decidable language.

Proof

M="On input <B,w>, where B is a DFA and w is a string:

1, Simulate B on input w;

2, If the simulation ends in an accept state, *accept*. If it ends in a nonaccepting state, *reject*."

Another decidable problem

A_{CFG}={<G,w>|G is a CFG that generates string w}

Theorem 4.7 A_{CFG} is a decidable language.

Proof

S="On input <G,w>

1, Convert G to an equivalent grammar in Chomsky normal form.

2, List all derivations with 2n-1 steps, where n is the length of w, except if n=0, then instead list all derivations with 1 step.

3, If any of these derivations generate w, *accept*; if not, *reject*."

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 ?

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

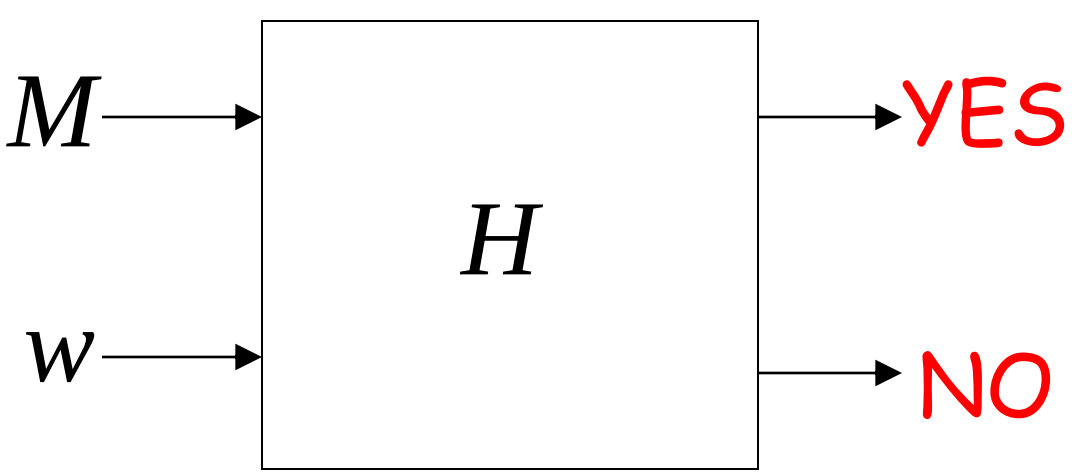
Theorem:

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



M accepts w

M rejects w

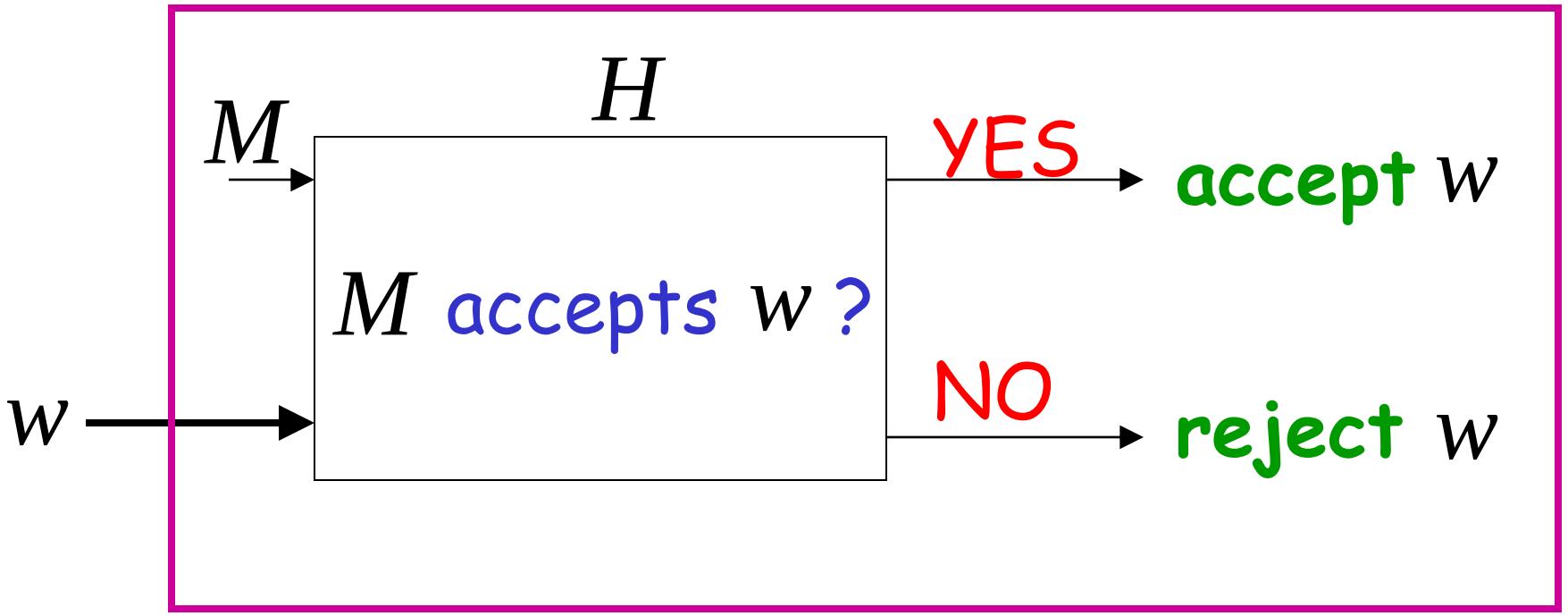
Let L be a recursively enumerable language

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



On w, M may loop because L is a recursively enumerable. But H always halts because H can decide <M, w>. So, on w, H halts. L is recursive

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

END OF PROOF

Another famous undecidable problem:

The halting problem

The Halting Problem

Input:

- Turing Machine M
- String w

Question: Does M halt on input w ?

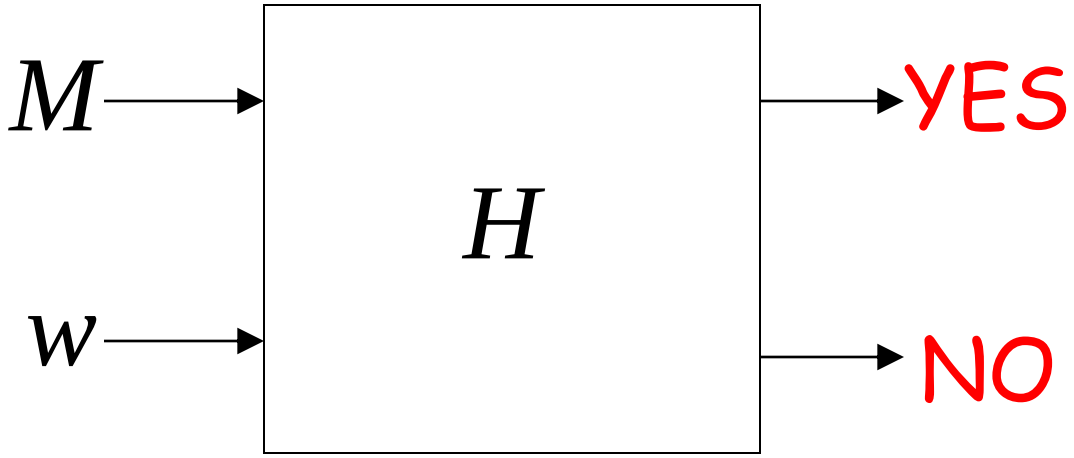
Theorem:

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

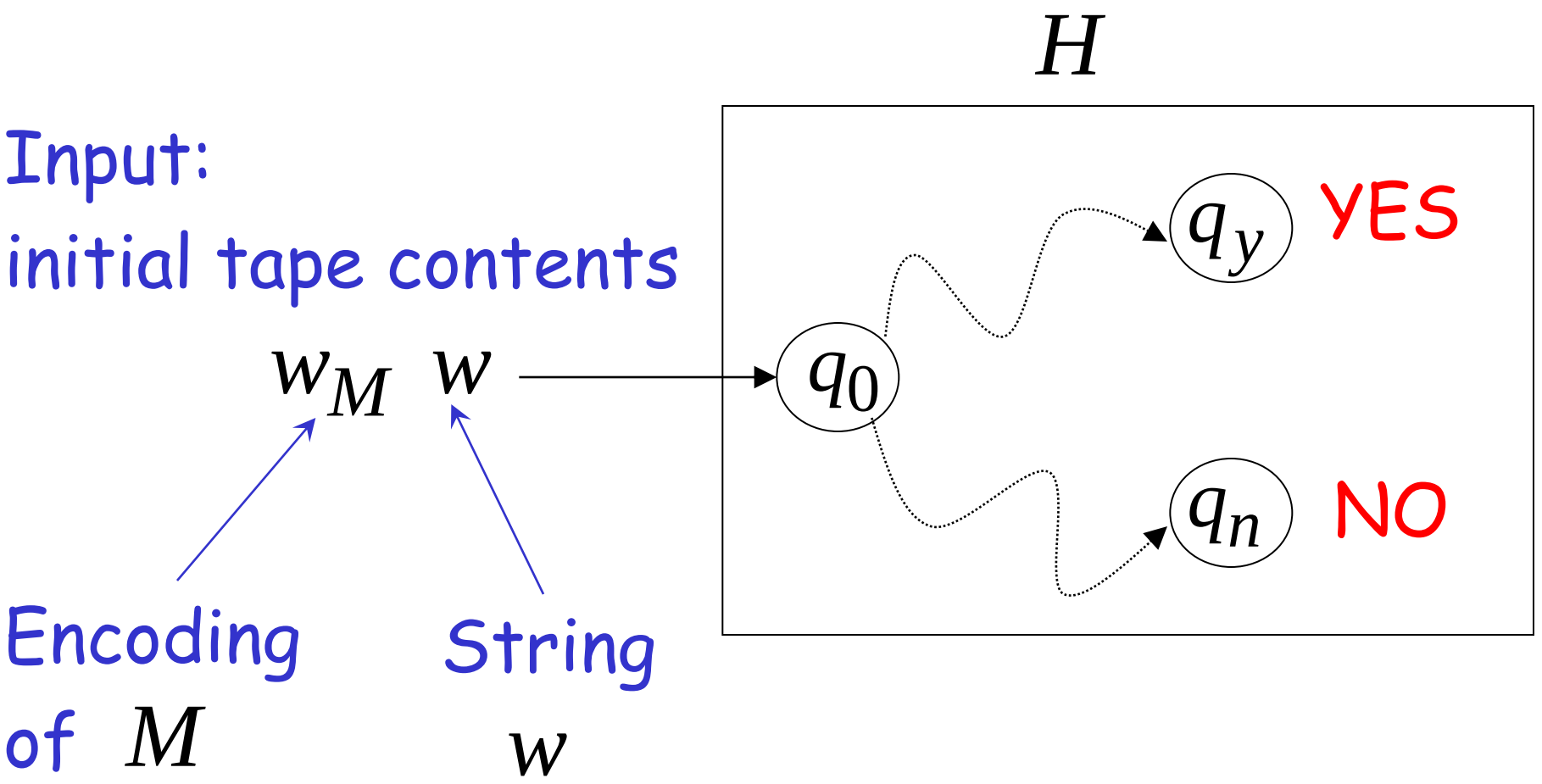
Thus, there exists Turing Machine H that solves the halting problem



M halts on w

M doesn't halt on w

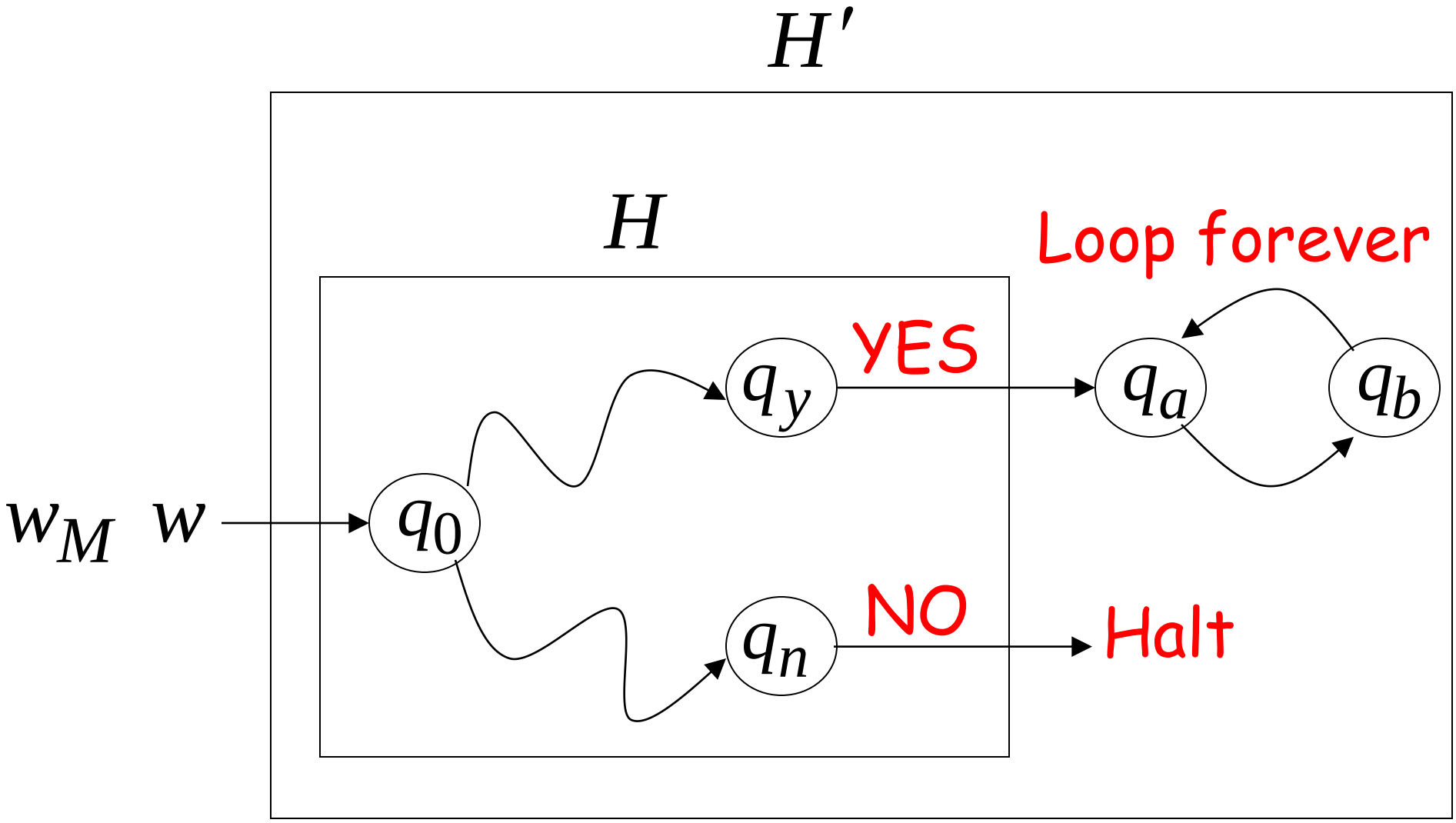
Construction of H



Construct machine H' :

If H returns YES then loop forever

If H returns NO then halt


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

Run machine H' with input itself:

Input: $w_{H'}$ (machine H')

If H' halts on input $w_{H'}$

Then loop forever

Else halt

H' on input $w_{H'}$:

If H' halts then loops forever

If H' doesn't halt then it halts

NONSENSE !!!!!

Therefore, we have contradiction

The halting problem is undecidable

END OF PROOF

Another proof of the same theorem:

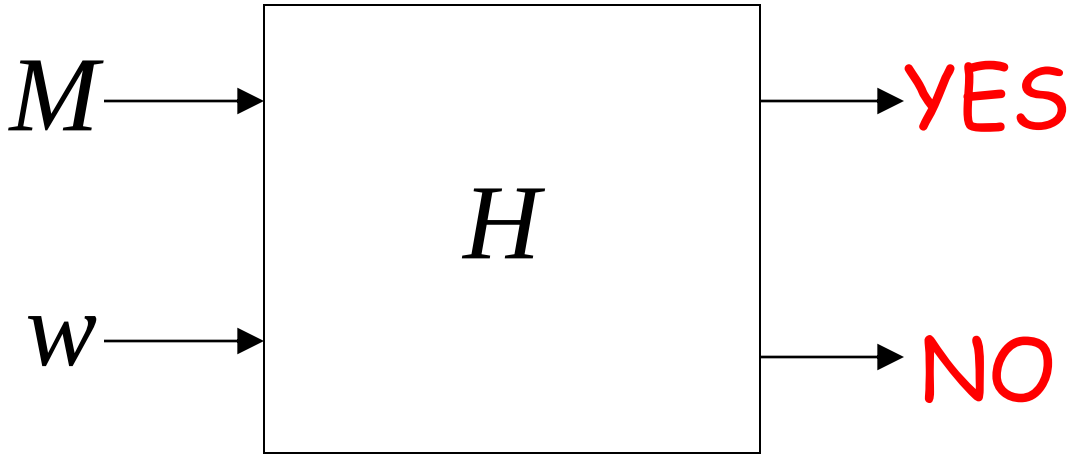
If the halting problem was decidable then every recursively enumerable language would be recursive

Theorem:

The halting problem is undecidable

Proof: Assume for contradiction that the halting problem is decidable

There exists Turing Machine H that solves the halting problem



M halts on w

M doesn't halt on w

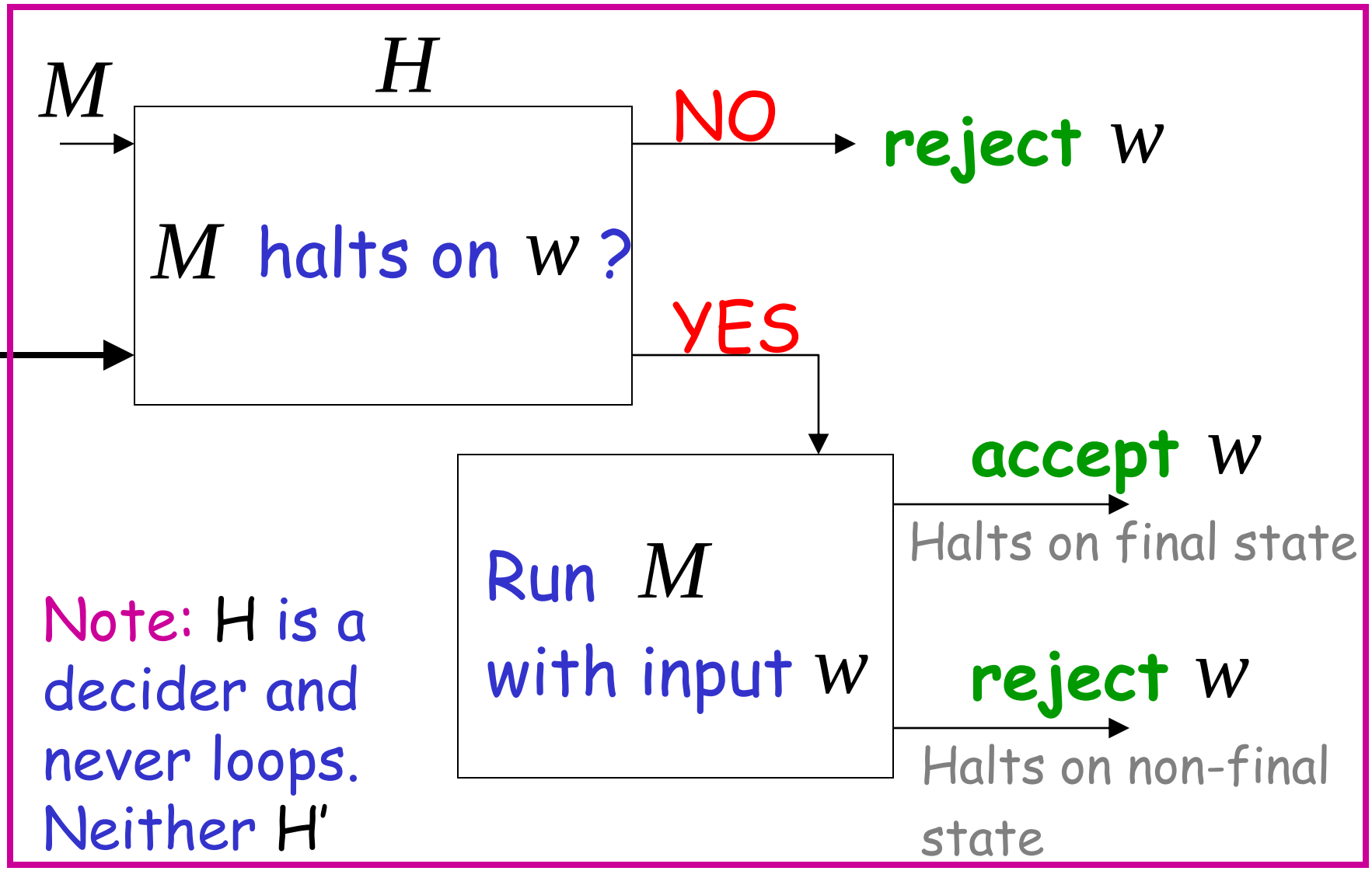
Let L be a recursively enumerable language

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

END OF PROOF

Another approach for understanding halting problem

The Diagonalization Method

- The diagonalization method was discovered by mathematician Georg Cantor in 1873.

- If we have two infinite sets, how can we tell whether one is larger or they are of the same size?

- we can't use the counting method to determine the relative sizes of infinite sets.

The Diagonalization Method

- Two finite sets have the same size if the elements of one set can be paired with the elements of the other set.

The Diagonalization Method

Definition 4.12

- Assume that sets A and B.
  - function f is one-to-one if it never maps two different elements to the same place.
  - f is onto if it hits every element of B.
- A function that is both one-to-one and onto is called a correspondence.

The Diagonalization Method

- A correspondence is simply a way of pairing the elements of A with the elements of B.
- Say that A and B are the same size if there is one-to-one, onto function f: $A \rightarrow B$.

The Diagonalization Method

Example 4.13

- Let N be the set of natural numbers {1, 2, 3,…} and let E be the set of even natural numbers {2, 4, 6,…}.
- N and E have the same size. The correspondence f mapping N to E is simple F(n)=2n.

n	f(n)
1	2
2	4
3	6
…	…

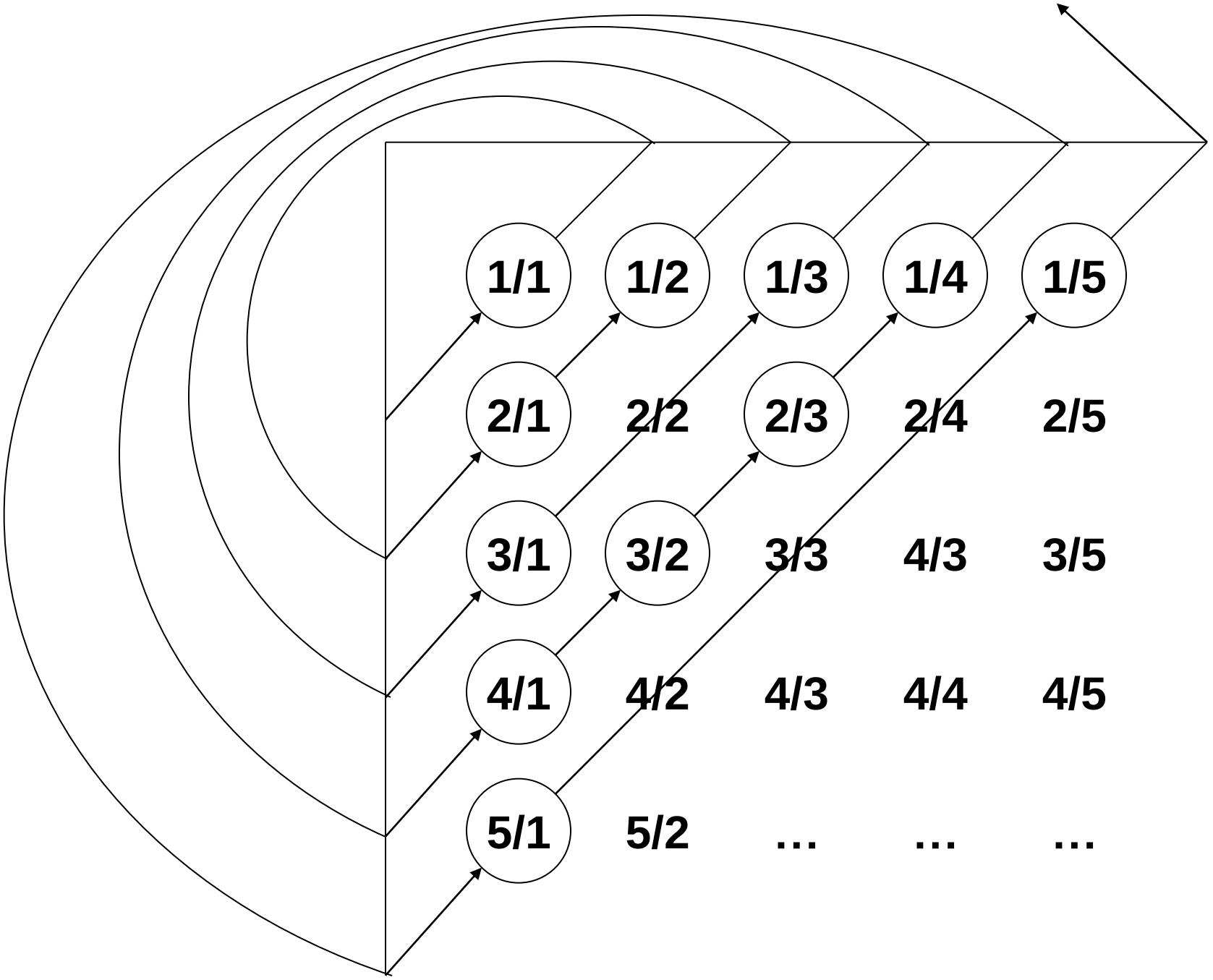
The Diagonalization Method

Definition 4.14

- A set A is countable if either it is finite or it has the same size as $\mathbb{N}$.

Example 4.15

- If we let $Q=\{m/n \mid m,n \in \mathbb{N}\}$ be the set of positive rational numbers, Q and $\mathbb{N}$ are the same size according.


1/1
1/2
1/3
1/4
1/5
2/1
2/2
2/3
2/4
2/5
3/1
3/2
3/3
4/3
3/5
4/1
4/2
4/3
4/4
4/5
5/1
5/2
…
…
…

The Diagonalization Method

- The set of real numbers is an example of an uncountable set.
- A real number is one that has a decimal representation.
- π=3.1415926… and $\sqrt{2}$ =1.4141135… are examples of real numbers. Let **R** be the set of real numbers.

The Diagonalization Method

- Theorem 4.17
  - R is uncountable
- Proof
  - Suppose that a correspondence f existed between N and R.
  - Let f(1)=3.14159, f(2)=55.55555…, f(3)=… and so on, just to make up some values for f. Then f pairs the number 1 with 3.14159…, the number 2 with 55.55555…, and so on.

The Diagonalization Method

Then we construct the desired x by giving its decimal representation.

n	f(n)
1	3.14159…
2	55.55555…
3	0.12345…
4	0.50000…
…	…

The Diagonalization Method

- we let the first digit of x be anything different from the first fractional digit of f(1)=3.14159…Arbitratarily, we let it be 4.
- we let the second fractional digit of x be anything different from the first fractional digit of f(2)=55.515555…Arbitratarily, we let it be 6. ……

The Diagonalization Method

We know that x is not f(n) for any n because it differs from f(n) in the n_{th} fractional digit.

n	f(n)
1	3.14159…
2	55.55555…
3	0.12345…
4	0.50000…
…	…

x=0.4641…

The Diagonalization Method

- Corollary 4.18 Some languages are not Turing-recognizable.
- We'll see that the set of all Turing machines is countable
- Proof
  - We have already proved that the set of all strings Σ^* is countable.

The Diagonalization Method

- Each Turing machine M has an encoding into a binary string <M>.
- The number of all Turing machines is a subset of a set of all string Σ^* .
- The set of all Turing machines is countable

The Diagonalization Method

- The set of all languages is same as a power set of an infinite countable set, which was proved to be uncountable;
- Thus we have shown that the set of all Turing machines can not be put into a correspondence with the set of all languages;
- We conclude that some languages are not recognized by any Turing machine.

The Halting Problem

- The problem of determining whether a Turing Machine accepts a given input string.

  $A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w\}$

- Proof
  - By contradiction, suppose that H is a decider for A_{TM}. On input $\langle M, w\rangle$, H halts and accepts if M accepts w.

The Halting Problem

$$\mathrm{H}(< M, w >) = \begin{cases} accept & \text{if M accepts w} \\ reject & \text{if M does not accept w} \end{cases}$$

- Now we construct a new Turing machine D with H as a subroutine.
- D determines what M does when the input to M is its own description <M>.
- Once D has determined this information, it does the opposite.

The Halting Problem

The following is a description of D.

D="ON input <M>, where M is a TM:

1. Run H on input <M, <M>>.

2. Output the opposite of what H outputs; that is, if H accepts, reject and if H rejects, accept."

The Halting Problem

In summary,

$$D(<M>) = \begin{cases} accept & \text{if M does not accept <M>} \\ reject & \text{if M accepts <M>} \end{cases}$$

	<M_1>	<M_2>	<M_3>	<M_4>	…
M_1	accept	reject	accept	reject	…
M_2	accept	accept	accept	accept	…
M_3	reject	reject	reject	reject	…
M_4	accept	accept	reject	reject	…
…	…	…	…	…	…

The Halting Problem

- There is no problem in the previous table. But what happens for D in next table? A contradiction occurs at "?"

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	…	$\langle D \rangle$	…
M_1	accept	reject	accept	reject	…	accept	…
M_2	accept	accept	accept	accept	…	accept	…
M_3	reject	reject	reject	reject	…	reject	…
M_4	accept	accept	reject	reject	…	accept	…
…	…	…	…	…	…	…	…
D	reject	reject	accept	accept	…	?	…
…	…	…	…	…	…	…	…

The Halting Problem

That means, when we run D with its own description <D> as input, we get

$$\mathrm{D}(< D >) = \begin{cases} accept & \text{if D does not accept <D>} \\ reject & \text{if D accepts <D>} \end{cases}$$

- No matter what D does, it is forced to do the opposite, which is obviously a contradiction. Thus neither TM D nor TM H can exist.

The Halting Problem

- Let's review the steps of this proof. Assume that a TM H decides A_{TM}. Then use H to build a TM D. The machines take the following actions:

H accepts <M, w> exactly when M accepts w.

D rejects <M> exactly when M accepts <M>.

D rejects <D> exactly when D accepts <D>.

The Halting Problem

- In summary, A_{TM} is not decidable. That means a TM can not decide whether a TM M would accept or reject a string w.
- Hence we can see that the ability of TM is still limited. That's the most important significance of the halting problem.