Undecidability
D. me(n log n)

D. me(n^2)

D. me(n log n)

D. me(n)

P

NP

NPC

EXP

RECURSIVE

R. E.

UNDECIDABLE

this & next lecture

not even accepted by a TM

this & next lecture

not even accepted by a TM
Recap: Typical TM code:

11101010000100100110100100000101011......11.......11.......111

- Begins, ends with 111
- Transitions separated by 11
- Fields within transition separated by 1
- Individual fields represented by 0s
Recap: Universal TM $M_u$

We saw a TM $M_u$ such that

$$L(M_u) = \{ <M> \# w \mid M \text{ accepts } w \}$$

Thus, $M_u$ is a stored-program computer.

It reads a program $<M>$ and executes it on data $w$

$$L_u = L(M_u) = \{ <M> \# w \mid M \text{ accepts } w \}$$ is r.e.
Recap: $L_u$ is not recursive

$\text{TM} \quad \text{Q}$

Copy-arg $\quad \langle M \rangle \# \langle M \rangle$

Decides if $M(\langle M \rangle)$ accepts

TM accept-checker

accept doesn’t

Q($\langle M \rangle$) rejects iff $M(\langle M \rangle)$ accepts
Q($\langle M \rangle$) accepts iff $M(\langle M \rangle)$ doesn’t accept

Does Q($\langle Q \rangle$) accept or reject?

either way, a contradiction, so assumption that accept-checker existed was wrong
UNDECIDABLE

R. E.

RECURSIVE

EXP

NP

P

Dtime(n)

Dtime(n log n)

Dtime(n^2)

Dtime(n^3)

not even accepted by a TM

this & next lecture

NPC

P

NP

EXP

R. E.

L_u

UNDECIDABLE
Polytime Reductions

\[ X \leq_p Y \quad \text{“}X\text{ reduces to } Y\text{ in polytime”} \]

If \( Y \) can be decided in poly time, then \( X \) can be decided in poly time

If \( X \) can’t be decided in poly time, then \( Y \) can’t be decided in poly time
Polytime-Reductions

$X \leq Y$ "$X$ reduces to $Y$ in polytime"

If $Y$ can be decided in poly-time, then $X$ can be decided in poly-time.

If $X$ can’t be decided in poly-time, then $Y$ can’t be decided in poly-time.
Reduction

$X \leq Y \quad \text{“}X \text{ reduces to } Y\text{”}$

If $Y$ can be decided, then $X$ can be decided.
If $X$ can’t be decided, then $Y$ can’t be decided.
Halting Problem

• Does given $M$ halt when run on blank input?
• $L_{halt} = \{<M> \mid M \text{ halts when run on blank input}\}$
• Show $L_{halt}$ is undecidable by showing $L_u \leq L_{halt}$

What are input and output of the reduction?
\[ L_u \leq L_{\text{halt}} \]

**REDUCTION**

**\( L_u \)-decider**

\[ <M> \ # \ w \rightarrow \text{REDUCTION} \rightarrow <M'> \rightarrow L_{\text{halt}} \text{ decider} \]

**YES**

**NO**

**Need:** \( M' \) halts on blank input iff \( M(w) \) accepts

\[
\text{TM } M' \\
\text{ const } M \\
\text{ const } w \\
\text{ run } M(w) \text{ and halt if it accepts}
\]

The REDUCTION **doesn’t run** \( M \) **on** \( w \). It produces code for \( M' \)!
$L_u \leq L_{\text{halt}}$

**Need:** $M'$ halts on blank input iff $M(w)$ accepts

**Correctness:** $L_u$-decider say “yes” iff $M'$ halts on blank input iff $M(w)$ accepts iff $<M>#w$ is in $L_u$
Who cares about halting TMs?

• Remember, TMs = programs
• Virtually all math conjectures can be expressed as a halting-TM question.

Example: Goldbach’s conjecture:

*Every even number > 2 is the sum of two primes.*
Program Goldbach

goldbach()

n = 4
WHILE is-sum-of-two-primes(n)
  n = n+2
HALT

is-sum-of-two-primes(n): boolean

FOR p ≤ q < n
  IF p, q, prime AND p+q=n THEN RETURN TRUE
RETURN FALSE

goldbach() halts iff Goldbach’s conjecture is false
Deciding mathematical truth

prove-theorem(T)

w = “”

WHILE NOT is-a-proof-of (w,T)

    w = lexicographically-next-string(w)

OUTPUT T + “is true”

prove-theorem(T) halts iff there is a proof of T.
More reductions about languages

• We’ll show other languages involving program behavior are undecidable:
  • $L_{0^{374}} = \{<M> \mid L(M) = \{0^{374}\}\}$
  • $L_{\neq \emptyset} = \{<M> \mid L(M) \text{ is nonempty}\}$
  • $L_{\text{pal}} = \{<M> \mid L(M) = \text{palindromes}\}$
  • many many others
\( L_{374} = \{ \langle M \rangle \mid L(M) = \{0^{374}\} \} \) is undecidable

- Given a TM \( M \), telling whether it accepts only the string \( 0^{374} \) is not possible
- Proved by showing \( L_u \leq L_{374} \)

**REDUCTION:** BUILD \( M' \)

\[ \langle M \rangle \# w \rightarrow \text{instance of } L_u \rightarrow \text{REDUCTION: BUILD } M' \rightarrow \langle M' \rangle = \text{instance of } L_{374} \]

What is \( L(M') \)?
- If \( M(w) \) accepts, \( L(M') = \{0^{374}\} \)
- If \( M(w) \) doesn’t accept, \( L(M') = \emptyset \)

**Q:** How does the reduction know whether or not \( M(w) \) accepts?

**A:** It doesn’t have to. It just builds (code for) \( M' \).
If there is a decider $M_{374}$ to tell if a TM accepts the language $\{0^{374}\}$...

Since $L_u$ is not decidable, $M_{374}$ doesn’t exist, and $L_{374}$ is undecidable
$L_{374} = \{<M> \mid L(M) = \{0^{374}\}\}$ is undecidable

• What about $L_{\text{accepts-374}} = \{<M> \mid M \text{ accepts } 0^{374}\}$

• Is this easier?
  – in fact, yes, since $L_{374}$ isn’t even r.e., but $L_{\text{accepts-374}}$ is
  – but no, $L_{\text{accepts-374}}$ is not decidable either

• The same reduction works:
  – If $M(w)$ accepts, $L(M') = \{0^{374}\}$, so $M'$ accepts $0^{374}$
  – If $M(w)$ doesn’t, $L(M') = \emptyset$, so $M'$ doesn’t accept $0^{374}$

• More generally, telling whether or not a machine accepts any fixed string is undecidable
$L_{\neq \emptyset} = \{<M> \mid L(M) \text{ is nonempty}\}$ is undecidable

- Given a TM $M$, telling whether it accepts any string is undecidable
- Proved by showing $L_u \leq L_{\neq \emptyset}$

**REDUCTION: BUILD $M'$**

We want $M'$ to satisfy:
- If $M(w)$ accepts, $L(M') \neq \emptyset$
- If $M(w)$ doesn’t $L(M') = \emptyset$

If $M(w)$ accepts, $L(M') = \Sigma^*$ hence $\neq \emptyset$
If $M(w)$ doesn’t, $L(M') = \emptyset$

On input $x$,
- Run $M(w)$
- Accept $x$ if $M(w)$ accepts
If there is a decider $M_{\neq \emptyset}$ to tell if a TM accepts a nonempty language...

Decider for $L_u$

REDUCTION: BUILD $M'$

$M'$: constants: $M$, $w$

On input $x$,
Run $M(w)$
Accept $x$ if $M(w)$ accepts

$M_{\neq \emptyset}$

YES:
$L(M') \neq \emptyset$
iff $M$ accepts $w$

NO:
$L(M') = \emptyset$
iff $M$ doesn't accept $w$

Since $L_u$ is not decidable, $M_{\neq \emptyset}$ doesn't exist, and $L_{\neq \emptyset}$ is undecidable
$L_{pal} = \{<M> \mid L(M) = \text{palindromes}\}$ is undecidable

- Given a TM $M$, telling whether it accepts the set of palindromes is undecidable
- Proved by showing $L_u \leq L_{pal}$

**Reduction: Build $M'$**

- Instance of $L_u$: $<M> \# w$
- Instance of $L_{pal}$: $<M'>$

We want $M'$ to satisfy:
- If $M(w)$ accepts, $L(M') = \{\text{palindromes}\}$
- If $M(w)$ doesn’t, $L(M') \neq \{\text{palindromes}\}$

**On input $x$,**

- Run $M(w)$
- Accept $x$ if $M(w)$ accepts and $x$ is a palindrome
If there is a decider $M_{\text{pal}}$ to tell if a TM accepts the set of palindromes

Decider for $L_u$

$<M>\#w$ $\rightarrow$ REDUCTION: BUILD $M'$ $\rightarrow$ $<M'>$ $\rightarrow$ $M_{\text{pal}}$  

YES: 
$L(M') = \{\text{palindromes}\}$ 
iff $M$ accepts $w$

NO: 
$L(M') = \emptyset \neq \{\text{palindromes}\}$ 
iff $M$ doesn’t accept $w$

$M'$: constants: $M$, $w$

On input $x$,

Run $M(w)$
Accept $x$ if
$M(w)$ accepts and
$x$ is a palindrome

Since $L_u$ is not decidable, $M_{\text{pal}}$ doesn’t exist, and $L_{\text{pal}}$ is undecidable
Lots of undecidable problems about languages accepted by programs

• Given $M$, is $L(M) = \{\text{palindromes}\}$?
• Given $M$, is $L(M) \neq \emptyset$?
• Given $M$, is $L(M) = \{0^{374}\}$?
• Given $M$, does $L(M)$ contain any prime?
• Given $M$, does $L(M)$ contain any word?
• Given $M$, does $L(M)$ meet these formal specs?
• Given $M$, does $L(M) = \Sigma^*$?
Rice’s Theorem

- Q: What can we decide about the languages accepted by programs?

A: NOTHING!

except “trivial” things
Properties of r.e. languages

- A **Property of r.e. languages** is a set of the form
  \( \{ \langle M \rangle \mid L(M) \text{ satisfies predicate } P \} \)
  where \( P \) is a predicate of r.e. languages.
  i.e., \( P: \{ L \mid L \text{ is r.e.} \} \rightarrow \{ \text{true, false} \} \)

- Examples:
  - \( P(L) = \text{“}L \text{ contains } 0^{374}\text{”} \)
  - \( P(L) = \text{“}L \text{ is empty}\text{”} \)
  - \( P(L) = \text{“}L = \{0^n1^n \mid n \geq 0\}\text{”} \)
Not properties of programs/TMs

• $P$ is defined on languages, not the machines which might accept them.
• $\{<M> \mid M \text{ at some point moves its head left}\}$ is a property of the machine behavior, not the language accepted.
• $\{<A.py> \mid \text{ program } A \text{ has 374 lines of code}\}$
• $\{<A.py> \mid \text{ A accepts TSP dec problem}\}$ this really is a predicate on $L(A)$
Trivial Properties

• \{<M> \mid L(M) \text{ is r.e}\}.... why is this “trivial” ?
  – EVERY language accepted by an \( M \) is r.e. by def’n

• \{<M> \mid L(M) \text{ is not r.e}\}.... why is this “trivial” ?

• \{<M> \mid L(M) = \emptyset \text{ or } L(M) \neq \emptyset\}.... why “trivial”? 

• A property is \textit{trivial} if either \textit{all} r.e. languages satisfy it, or \textit{no} r.e. languages satisfy it.

• Clearly, trivial properties are decidable
Rice’s Theorem

Every nontrivial property of r.e. languages is undecidable

So, there is virtually nothing we can decide about behavior (language accepted) by programs

Example: auto-graders don’t exist (if submissions are allowed to run an arbitrary (but finite) amount of time).
Proof

• Let $P$ be a non-trivial property
• Let $L_p = \{ <M> \mid L(M) \text{ satisfies predicate } P \}$
• Show $L_p$ is undecidable
• Assume $\emptyset$ does not satisfy $P$
• Assume $L(M_{p\text{-sat}})$ satisfies $P$ for some TM $M_{p\text{-sat}}$

There must be at least one such TM (why?)
If there is a decider $M_p$ to tell if a TM accepts a language satisfying $P$...

Decider for $L_u$

REDUCTION: BUILD $M'$

$M'$: constants: $M$, $w$

On input $x$,
- Run $M(w)$
- Accept $x$ if $M(w)$ accepts and $M_{P-sat}(w)$ accepts

YES: $L(M')$ satisfies $P$ iff $M$ accepts $w$

NO: $L(M') = \emptyset$ doesn’t satisfy $P$ iff $M$ doesn’t accept $w$

If $M$ doesn’t accept $w$ then $L(M') = \emptyset$
If $M$ does accept $w$ then $L(M') = L(M_{P-sat})$

Since $L_u$ is not decidable, $M_p$ doesn’t exist, and $L_p$ is undecidable
What about assumption

• We assumed $\emptyset$ does not satisfy $P$
• What if $\emptyset$ does satisfy $P$?
• Then consider
  $$L_{p'} = \{ <M> \mid L(M) \text{ doesn't satisfy predicate } P \}$$
• Then $\emptyset$ isn’t in $L_{p'}$
• Show $L_{p'}$ is undecidable
• So $L_p$ isn’t either (by closure under complement)
Properties about TMs

- sometimes decidable:
  - \{<M> \mid M \text{ has } 374 \text{ states}\}
  - \{<M> \mid M \text{ uses } \leq \ 374 \text{ tape cells on blank input}\}
    - \ 374 \times |\Gamma|^{32} \times |Q_M|
  - \{<M> \mid M \text{ never moves head to left}\}

- sometimes undecidable
  - \{<M> \mid M \text{ halts on blank input}\}
  - \{<M> \mid M, \text{ on input “0110”, eventually writes “2”}\}
Today

• Quick recap – halting & undecidability
• r.e. and non-r.e. languages
• ICES
  – pick up TWO forms (Chandra + Lenny)
  – return to same location
\[ \text{D.me(n)} \]
\[ \text{D.me(n \log n)} \]
\[ \text{D.me(n^2)} \]
\[ \text{D.me(n^3)} \]

\[ \text{P} \]
\[ \text{NP} \]
\[ \text{NPC} \]

\[ \text{EXP} \]
\[ \text{RECURSIVE} \]

\[ \text{UNDECIDABLE} \]

\[ \text{not even accepted by a TM} \]

\[ \text{this \& next lecture} \]
Recall Halting Problem:

• Given $<M>$, does $M$ halt when run on no input?
• We showed this was undecidable.
• We pointed out consequences:
• If we could determine halting, we could solve many many problems whose solution is unknown
Program Goldbach

goldbach()

n = 4

WHILE is-sum-of-two-primes(n)

    n = n+2

HALT

is-sum-of-two-primes(n): boolean

FOR p ≤ q < n

    IF p, q, prime AND p+q=n THEN RETURN TRUE

RETURN FALSE

goldbach() halts iff Goldbach’s conjecture is false
CS 125 assignment:

• Write a program that outputs “Hello world”.

```
main()
{
    printf("Hello world");
}
```

• Can we write an auto-grader?

• If so; we can solve Goldbach’s conjecture...
goldbach()
  
n = 4
  
  WHILE is-sum-of-two-primes(n)
    
    n = n+2
  
  HALT

is-sum-of-two-primes(n): boolean
  
  FOR p ≤ q < n
    
    IF p, q, prime AND p+q=n

    THEN RETURN TRUE
  
  RETURN FALSE

main()
  
  { goldbach();
    printf(“Hello world”);
  }

AUTOGRADER

CORRECT

INCORRECT
Deciding halting problem

- Given program \(<M>\), to determine if \(M\) halts, do the following:

  So, deciding if a program prints “Hello world” is solving the halting problem

Using same ideas, we can show that deciding anything about code behavior is not possible
Non-r.e. languages
non-rec. vs non-r.e.

• A language is not recursive (not decidable) iff there is no always-halting program to decide whether or not strings are in the language.

• A language is not r.e. iff there is not any program that accepts exactly the set of strings in the language (even if it may fail to halt on strings not in the language).
Intuition – r.e.

• A language is r.e. if a finite search can verify that a string is in the language (search may be infinite if the string is not) [note “verify” is analogous to NP]
  – \{<M> | L(M) is nonempty\}
    (just dovetail \(M(w)\) for all strings \(w\) and accept if one is found that \(M\) accepts)
  – \{<M> | M halts on blank input\}
    (just run \(M\) and accept if it halts)
  – \{n | \(\pi\) contains the substring \(a7^n b\) such that \(a, b \neq 7\)\}
    (just enumerate digits of \(\pi\) and accept if \(a7^n b\) is found)
Intuition – non-r.e.

• A language is not even r.e. if no finite search can verify a string is in the language.
  – \{<M> | L(M) is empty\}
    (what search could convince you that no string is accepted by \(M\)?)
  – \{<M> | L(M) is infinite\}
    (how can you check this in finite time?)
  – \{<M> | L(M) = \(\Sigma^*\), or \{<M> | L(M) \(\neq\) \(\Sigma^*\)\}

Note: these are not proofs of non-r.e.; only intuitions
Test your intuition
answer recursive, r.e., or not r.e.

• \{<M> \mid L(M) \text{ is finite}\}
• \{<M> \mid L(M) \text{ contains some prime number}\}
• \{<M> \mid L(M) \text{ is nonempty}\}
• \{<M> \mid L(M) \text{ is empty}\}
• \{<M>\#w \mid M(w) \text{ prints a finite number of 1s}\}
• \{<M>\#w \mid M(w) \text{ ever moves left three times successively}\}
Proving a language non-r.e.

- counting argument
  - proves existence of non-r.e. languages
- diagonalization
  - construct a particular non-r.e. language $L_d$
- show complement($L$) is r.e. but not recursive
- reduce a known non-r.e. language to $L$
Counting Argument

Following 7 slides from first lecture prove that there are non-r.e. languages, since the number of TMs, or programs, is only countably infinite, while the number of languages is uncountably infinite.
A non-computability proof

There are too many functions to compute

- Simple counting argument showing that we do not have enough algorithms to compute all of the boolean functions.
- There are just not enough algorithms to go around – most things are not computable.
Recall from CS 173

• $N = \text{natural numbers are countably infinite}$, meaning it is possible to enumerate them on a (neverending) list that contains each element

• $R = \text{real numbers are NOT countably infinite}$. Cantor’s *diagonalization proof* showed that $R$ cannot be put into one-to-one correspondence with $N$.

• In short: $|N| \ll |R|$
**Boolean functions on $N$**

- Let $F= \{f : f$ is a function from $N$ to $\{0,1\}\}$.
  - Thus, functions in $F$ map natural numbers to $\{0,1\}$.
  - $F$ contains, for example:
    
    $f(n) = n \mod 2$
    $f(n) = 0$ if $n$ is composite, $1$ if $n$ is prime
    $f(n) = 1$ iff $n$ is the binary file for a Python program that cannot get into an infinite loop on any input
    etc.
How large is $F$?

Each function $f$ in $F$ can be represented as an infinitely long bitstring, giving the values in order for $f(0)f(1)f(2)\ldots$

Example: $f(n) = n \mod 2$:
then represent $f$ by “0101010101\ldots”

Example: $f(n) = 0$ if $n$ composite, 1 if prime
then represent $f$ by “00110101000101\ldots”
Each $f$ corresponds to a real $r$ in $[0,1]$

Just put a decimal point in front of $f$’s sequence and interpret as a real number

- $f = "010101..."$ corresponds to $0.0101...$
- $f = "10000..."$ corresponds to $0.1 = \frac{1}{2}$
- etc.

And every $r$ in $[0,1]$ corresponds to some $f$.

So, there are just as many functions in $F$ as there are elements of $[0,1] = \text{cardinality of the continuum, } = |\mathbb{R}|$, which is uncountably infinite
How many programs are there?

- What do we mean by program?
- Finite length code, over finite set of symbols
  (e.g., a-z; A-Z; 0-9, “;”, ./,\,?,>,<,,{},(,),...)
  Let’s assume alphabet of 256 possible characters
- Too hard to count valid programs, let’s overcount by just counting finite character sequences:
  - 256 of length 1  (= a, b, c, d, ...)
  - $256^2$ of length 2  (= aa, ab, ac, ..., az, ba, bb, ...)
  - $256^3$ of length 3
  etc.
Only countably many programs!

- List all of length 1, of length 2, of length 3, ... gives an enumeration of all valid programs (and many invalid ones).
- Because they can be enumerated, ...
  \[ |\{\text{programs}\}| = |N| \ll |R| = |F| \]
- In fact, the *vast majority* of functions in $F$ have no program computing them.
- We’ll see many specific ones later in course, and learn how to prove things *undecidable*. 
Diagonalization

• Construct a particular non-r.e. language
  <construction had better be non-algorithmic>
• Recall TMs are numbered 0,1,2,3,...
• Let $\Sigma = \{0,1\}$, and let $w_0, w_1, w_2, ...$ be a lexicographic enumeration of strings of $\Sigma^*$
• Then each language $L(M_i)$ can be expressed as a row in an infinite table, telling whether each $w_j$ is in $L(M_i)$ or not...
List of all r.e. languages

<table>
<thead>
<tr>
<th></th>
<th>$w_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$w_5$</th>
<th>$w_6$</th>
<th>$w_7$</th>
<th>$w_8$</th>
<th>$w_9$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_0$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$M_1$</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$M_2$</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$M_3$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>$M_4$</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$M_5$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$M_6$</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>$M_7$</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>$M_8$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$M_9$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### List of all r.e. languages

<table>
<thead>
<tr>
<th></th>
<th>$w_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$w_5$</th>
<th>$w_6$</th>
<th>$w_7$</th>
<th>$w_8$</th>
<th>$w_9$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_0$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_1$</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_2$</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_3$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>...</td>
</tr>
<tr>
<td>$M_4$</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_5$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_6$</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>...</td>
</tr>
<tr>
<td>$M_7$</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>...</td>
</tr>
<tr>
<td>$M_8$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_9$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Consider for each $i$, whether or not $M_i$ accepts $w_i$.
List of all r.e. languages

<table>
<thead>
<tr>
<th></th>
<th>$w_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$w_5$</th>
<th>$w_6$</th>
<th>$w_7$</th>
<th>$w_8$</th>
<th>$w_9$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_0$</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_1$</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_2$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_3$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>...</td>
</tr>
<tr>
<td>$M_4$</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_5$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_6$</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>...</td>
</tr>
<tr>
<td>$M_7$</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>...</td>
</tr>
<tr>
<td>$M_8$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>$M_9$</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Flip “yes” and “no”, defining $L_d = \{w_i \mid w_i \text{ not in } L(M_i)\}$
\[ L_d = \{ w_i \mid w_i \text{ not in } L(M_i) \} \]

\( L_d \) is not r.e.

- if it were, it would be accepted by some TM \( M_k \)
- but \( L_d \) contains \( w_k \) iff \( L(M_k) \) does not contain \( w_k \)
- so \( L_d \neq L(M_k) \) for any \( k \)
- so \( L_d \) is not r.e.
Showing complement(L) is r.e. but not recursive

Theorem
L is recursive iff L and L' = complement(L) are both r.e.

Proof
(only if) If L recursive, then so is L', and both are r.e.
(if) If L and L' are both r.e., then M decides whether or not w in L by dovetailing M_L(w) and M_L'(w) until one accepts, and then accepts or rejects accordingly.
Corollary

If $L'$ is r.e. but not recursive, then $L$ is not r.e.

Example

- $L_{\neq \emptyset} = \{ <M> \mid L(M) \text{ nonempty} \}$ was shown non-recursive.
- Easy to see that $L_{\neq \emptyset}$ is r.e. (how?)
- Thus $L_{= \emptyset} = \{ <M> \mid L(M) \text{ is empty} \}$ is not r.e.
  - (matches intuition.... how could you ever verify?)
D.m(e(n)

D.m(e(n

D.m(e(n

D.m(e(n

P

NP

NPC

EXP

RECURSIVE

R. E.

UNDECIDABLE

SUMMARY

not even accepted by a TM