Turing Machine Recap
• DFA with (infinite) tape.
• One move: read, write, move, change state.
High-level Points

• **Church-Turing thesis:** TMs are the most general computing devices. So far no counter example

• **Every TM can be represented as a string.** Think of TM as a program but in a very low-level language.

• **Universal Turing Machine** $M_u$ that can simulate a given $M$ on a given string $w$
Decision Problems

- A yes/no question over many instances
  - Given grammar G, is G ambiguous?
  - Given a TM M, does \( L(M) = \{0,1\}^* \)?
  - Given DFAs \( M_1 \) and \( M_2 \), does \( L(M_1) = L(M_2) \)?
  - Given a graph G, is G connected?
  - Given a graph G, nodes s and t, and number d, is there a path from s to t of distance d or less?
Equivalently, languages:

- \{<G> \mid <G> \text{ encodes an unambiguous grammar}\}
- \{<M> \mid L(M) = \{0,1\}^*\}
- \{<M_1> \# <M_2> \mid \text{DFAs } M_1 \text{ and } M_2, \text{ accept the same language}\}
- \{<G> \mid <G> \text{ encodes a connected graph}\}
- \{<G>\#s\#t\#d \mid <G> \text{ encodes a graph with nodes } s \text{ and } t, \text{ there is a path from } s \text{ to } t \text{ of distance } d \text{ or less}\}

Deciding membership in the language is solving the decision problem
Decidable

• A decision problem (language) is *decidable* if there is a TM that always halts that accepts the language. (The language is recursive.)

• I.e., there is an algorithm that always answers “yes” or “no” correctly.

• Note: since all finite languages are recursive, (they’re regular in fact) any decision problem with only a finite number of instances is decidable, and not well-addressed by this theory....
Example 1: decidable or not?

• Is there a substring of exactly 374 consecutive 7’s in decimal expansion of $\pi$?

• This is decidable. There is an algorithm which is correct. It is one of these:

  - Alg 1: Output "yes"
  - Alg 2: Output "no"

We just don’t know which one it is. But, there is an algorithm which will tell us which it is!
Moral

• This is nonsense
• There were no “instances” of the problem.
• It simply asks a single yes/no question.
• Not even clear what “language” corresponds to it
• Remember: decidability is for problems with many possible input instances
Example 2

• Give $n$, is there a substring of exactly $n$ consecutive 7’s in $\pi$?
• Language: $\{n \mid$ decimal expansion of $\pi$ contains the substring $a7^nb$, where $a$ and $b$ are not 7s$\}$
• Is this language decidable? Is there a halting TM for it?
• Is it r.e.? (recall: a TM that may not halt but accepts if it should)
Example 3

• Give \( n \), is there a substring of \( \text{at least } n \) consecutive 7’s in \( \pi \) ?

• Language: \( L = \{ n \mid \text{decimal expansion of } \pi \text{ contains the substring } 7^n \} \)

• Is this language decidable? Is there a halting TM for it?

• In fact, it is regular!

  (\( L \) is either all of \( \mathbb{N} \), or equals \( \{0,1,2,...,k\} \) for some fixed \( k \).)
Universal TM

• A single TM $M_u$ that can compute anything computable!
• Takes as input
  – the \textit{description} of some other TM $M$
  – data $w$ for $M$ to run on
• Outputs
  – the results of running $M(w)$
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
- Note: this can be viewed as a natural number
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 $w$

$$L_u = L(M_u) = \{ <M> \# w \mid M \text{ accepts } w \} \text{ is r.e.}$$
High-level Points

• **Church-Turing thesis:** TMs are the most general computing devices. So far no counter example

• **Every TM can be represented as a string.** Think of TM as a program but in a very low-level language.

• **Universal Turing Machine** $M_u$ that can simulate a given $M$ on a given string $w$
Undecidability
Dtime(n)
Dtime(n log n)
Dtime(n^2)
Dtime(n^3)

P
NP
NPC

EXP

RECURSIVE

R. E.

UNDECIDABLE

this lecture

not even accepted by a TM
Undecidable Languages: Counting Argument

• Are there undecidable languages?
• Most languages are undecidable!
• Simple proof:
  – # of TMs/algorithms is countably infinite since each TM can be represented as a natural number (it’s description is a unique binary number)
  – # of languages is uncountably infinite
Is $L_u$ decidable?

- Counting argument does not directly tell us about undecidability of specific interesting languages
- Recall $L_u = \{ <M>\#w \mid M \text{ accepts } w \}$ is r.e.
- Is $L_u$ decidable?
Halting Problem

• Does given $M$ halt when run on blank input?
• $L_{halt} = \{<M> \mid M \text{ halts when run on blank input}\}$
• Is $L_{halt}$ decidable?
Who cares about halting TMs?
Who cares about halting TMs?

• Remember, TMs = programs
• Debugging is an important problem in CS
• Furthermore, 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

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()

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

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

• Write a program that outputs “Hello world”.

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

• Can you write an auto-grader?

• If so; you 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");
}

So, deciding if a program prints “Hello world” is solving goldbach’s conjecture
Deciding halting problem

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

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

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.
$L_u$ is not recursive

Two proofs

- Slick proof
- Slow proof via diagonalization and reduction
$L_u$ is not decidable

Warm-up: Self-reference leads to paradox

- In a town there is a barber who shaves all and only those who do not shave themselves
  
  **Who shaves the barber?**

- Homogenous words: self-describing
  - English, short, polysyllabic
  Heterogenous words: non-self-describing
  - Spanish, long, monosyllabic

What kind of word is “heterogenous”?
$L_u$ is not decidable

- Proof by contradiction
- Suppose there was an algorithm (TM) that always halted, as follows:

  $$<M> \# w \rightarrow \text{TM accept-checker} \rightarrow \text{Check if } M(w) \text{ accepts}$$

  yes, $M(w)$ accepts

  no, $M(w)$ doesn’t accept*

* remember – $M(w)$ may not halt – which is why this may be difficult

We’ll show how to use this as a subroutine to get a contradiction
$L_u$ is not decidable

- Proof by contradiction
- Suppose there was an algorithm (TM) as follows:

$$
\text{TM accept-checker}
$$

Decides if $M(<M>)$ accepts

$Q(<M>)$ accepts iff $M(<M>)$ doesn’t accept

$Q(<M>)$ rejects iff $M(<M>)$ accepts
$L_u$ is not decidable

TM Q

TM accept-checker
Decides if $M(<M>)$ accepts

$Q(<M>)$ accepts iff $M(<M>)$ doesn’t accept
$Q(<M>)$ rejects iff $M(<M>)$ accepts

Does $Q(<Q>)$ accept or reject?

either way, a contradiction, so assumption that accept-checker existed was wrong
$L_u$ is not decidable: Slow proof

- Use diagonalization to prove that a specific language $L_d$ is not r.e
- Show that if $L_u$ is decidable then $L_d$ is decidable which leads to contradiction
Diagonalization

• Fix alphabet to be \{0,1\}
• Recall that \{0,1\}^* is countable: we can enumerate strings as \(w_0, w_1, w_2, \ldots\)
• Recall that we established a correspondence between TMs and binary numbers hence TMs can be enumerated as \(M_0, M_1, M_2, \ldots\)
• A language \(L\) is a subset of \(\{0,1\}^*\)
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 (\cdots) no (\cdots) no (\cdots) no (\cdots) no (\cdots) no (\cdots) no (\cdots) no (\cdots) no (\cdots) no (\cdots) ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M_1)</td>
<td>yes (\cdots) no (\cdots) no (\cdots) yes (\cdots) no (\cdots) yes (\cdots) yes (\cdots) yes (\cdots) no (\cdots) ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M_2)</td>
<td>no (\cdots) yes (\cdots) yes (\cdots) no (\cdots) no (\cdots) yes (\cdots) no (\cdots) yes (\cdots) no (\cdots) no (\cdots) ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M_3)</td>
<td>no (\cdots) yes (\cdots) no (\cdots) yes (\cdots) no (\cdots) yes (\cdots) no (\cdots) yes (\cdots) yes (\cdots) no (\cdots) ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M_4)</td>
<td>yes (\cdots) yes (\cdots) yes (\cdots) yes (\cdots) no (\cdots) no (\cdots) no (\cdots) no (\cdots) no (\cdots) no (\cdots) ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M_5)</td>
<td>no (\cdots) no (\cdots) no (\cdots) no (\cdots) no (\cdots) no (\cdots) no (\cdots) no (\cdots) no (\cdots) no (\cdots) ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M_6)</td>
<td>yes (\cdots) yes (\cdots) yes (\cdots) yes (\cdots) yes (\cdots) yes (\cdots) yes (\cdots) yes (\cdots) yes (\cdots) yes (\cdots) ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M_7)</td>
<td>yes (\cdots) yes (\cdots) no (\cdots) no (\cdots) yes (\cdots) yes (\cdots) yes (\cdots) no (\cdots) no (\cdots) yes (\cdots) ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M_8)</td>
<td>no (\cdots) yes (\cdots) no (\cdots) no (\cdots) yes (\cdots) no (\cdots) yes (\cdots) yes (\cdots) yes (\cdots) no (\cdots) ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M_9)</td>
<td>no (\cdots) no (\cdots) no (\cdots) yes (\cdots) yes (\cdots) no (\cdots) yes (\cdots) no (\cdots) yes (\cdots) yes (\cdots) ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
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>yes</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>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>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>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>yes</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>no</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>no</td>
<td>no</td>
<td>no</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>no</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>no</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>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>yes</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>no</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>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>yes</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>no</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>

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.  (Why not?)

\begin{itemize}
\item if it were, it would be accepted by some TM \( M_k \)
\item but \( L_d \) contains \( w_k \) iff \( L(M_k) \) does not contain \( w_k \)
\item so \( L_d \neq L(M_k) \) for any \( k \)
\item so \( L_d \) is not r.e.
\end{itemize}
Reduction

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

If Y can be decided, then X can be decided.
If X can’t be decided, then Y can’t be decided.
$L_d \leq \overline{L_u}$

$L_d$-decider

REDUCTION $\rightarrow$ <$M, w$> $\rightarrow$ $\overline{L_u}$ decider

- YES: $M(w)$ doesn’t accept
- NO: $M(w)$ does accept
$L_d \leq \overline{L_u}$

$L_d$-decider

$w_i \xrightarrow{\text{REDUCTION}} <M_i, w_i> \xrightarrow{\overline{L_u} \text{ decider}}$

- YES $M_i(w_i)$ doesn’t accept
- NO $M_i(w_i)$ does accept

- The above is a reduction from $L_d$ to complement of $L_u$
- Note that a language $L$ is decidable iff $\overline{L}$ is decidable
- Hence $L_u$ is decidable iff $\overline{L_u}$ decidable
$L_u$ is not decidable

- $L_d$ is not r.e. by diagonalization
- Suppose $L_u$ is decidable
- Then $\overline{L_u}$ is also decidable
- We have shown $L_d \leq \overline{L_u}$ which implies $L_d$ is decidable, a contradiction
- Therefore $L_u$ is not decidable (undecidable)
- No algorithm for $L_u$
Using Reductions

• Once we have some seed problems such as \( L_d \) and \( L_u \) we can use reductions to prove that more problems are undecidable
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?
\[ L_u \leq L_{halt} \]

**L_u-decider**

\[ \langle M \rangle \# w \]

REDUCTION \[ \langle M' \rangle \]

\[ L_{halt} \text{ decider} \]

**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 \( M' \)!
Example

• Suppose we have the code for a program `isprime()` and we want to check if it accepts the number 13
• The reduction creates new program to give to decider for $L_{\text{halt}}$: note that the reduction only creates the code, does not run any program itself.

```c
main() {
    If (isprime(13)) then
        HALT
    else
        LOOP FOREVER
}

boolean isprime(int i) {
    ...
}
```
\[ L_u \leq L_{halt} \]

**\( L_u \)-decider**

\[ \langle M \rangle \# w \rightarrow \text{REDUCTION} \rightarrow \langle M' \rangle \rightarrow L_{halt} \text{ decider} \]

**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 \( \langle M \rangle \# w \) is in \( L_u \)
More reductions about languages

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

\[<M> \neq w \rightarrow \text{instance of } L_u\]

\[<M'> = \text{instance of } L_{374}\]

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

**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 \( \{0^{374}\} \)...
\[ L_{374} = \{ <M> \mid L(M) = \{0^{374}\} \} \text{ is undecidable} \]

- What about \( L_{accepts-374} = \{ <M> \mid M \text{ accepts } 0^{374} \} \)?
  - In fact, yes, since \( L_{374} \) isn’t even r.e., but \( L_{accepts-374} \) is.
  - But no, \( L_{accepts-374} \) 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 \textit{any} string is undecidable
- Proved by showing $L_u \leq L_{\neq \emptyset}$

\[<M> \# w\]

\[\text{instance of } L_u\]

\[\text{REDUCTION: BUILD } M'\]

\[<M'> = <M>\]

\[\text{instance of } L_{\neq \emptyset}\]

\[M' : \text{ constants: } M, w\]

On input $x$,

\[\text{Run } M(w)\]

Accept $x$ if $M(w)$ accepts

We want $M'$ to satisfy:

- If $M(w)$ accepts, $L(M') \neq \emptyset$
- If $M(w)$ doesn't, $L(M') = \emptyset$

What is $L(M')$?

- If $M(w)$ accepts, $L(M') = \Sigma^*$ hence $\neq \emptyset$
- If $M(w)$ doesn't, $L(M') = \emptyset$
If there is a decider $M \neq \emptyset$ to tell if a TM accepts a nonempty language...

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’**

<table>
<thead>
<tr>
<th>(&lt;M&gt; # w)</th>
<th>instance of (L_u)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;M’&gt;)</td>
<td>instance of (L_{pal})</td>
</tr>
</tbody>
</table>

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 $L_u$

REDUCTION: BUILD $M'$

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

On input $x$,

Run $M(w)$

Accept $x$ if

$M(w)$ accepts and $x$ is a palindrome

YES:

$L(M') = \{\text{palindromes}\}$

iff $M$ accepts $w$

NO:

$L(M') = \emptyset \neq \{\text{palindromes}\}$

iff $M$ doesn’t accept $w$

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 $0^{374}$?
- Given $M$, is $L(M) = \{\text{prime numbers}\}$?
- 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 $L(M) = \Sigma^*$?
Dtime(n)

Dtime(n log n)

Dtime(n^2)

Dtime(n^3)

P

NP

NPC

EXP

RECURSIVE

R. E.

UNDECIDABLE

not even accepted by a TM

SUMMARY