Foundations of Computer Science
Lecture 27

Unsolvable Problems

No Automatic Program Verifier for Hello-World
No Ultimate Debugger or Algorithm for PCP
The Complexity Zoo
Intuitive notion of algorithm ≡ Turing Machine
Solvable problem ≡ Turing-decidable

\[ \mathcal{L} = \{ \langle G \rangle \mid G \text{ is connected} \} \]

\[ \langle G \rangle = 2; 1; 3; 4 \ # 1,2; 2,3; 1,3; 3,4 \]

(\( \langle G \rangle \) is the encoding of graph \( G \) as a string.)
Last Time: Turing Machines

Intuitive notion of algorithm \equiv Turing Machine

Solvable problem \equiv Turing-decidable

\[ \mathcal{L} = \{ \langle G \rangle \mid G \text{ is connected} \} \]

\[ \langle G \rangle = 2; 1; 3; 4 \# 1,2; 2,3; 1,3; 3,4 \]

(\langle G \rangle \text{ is the encoding of graph } G \text{ as a string.})

\[
M = \text{Turing Machine that solves graph connectivity} \\
\text{input: } \langle G \rangle, \text{ the encoding of a graph } G.
\]
Last Time: Turing Machines

Intuitive notion of algorithm \equiv Turing Machine
Solvable problem \equiv Turing-decidable

\[ \mathcal{L} = \{ \langle G \rangle \mid G \text{ is connected} \} \]

\[ \langle G \rangle = 2; 1; 3; 4 \; \# \; 1,2; 2,3; 1,3; 3,4 \]

(\langle G \rangle \text{ is the encoding of graph } G \text{ as a string.})

\[ M = \text{Turing Machine that solves graph connectivity} \]

**input:** \langle G \rangle, the encoding of a graph \( G \).
1. Check that \langle G \rangle is a valid encoding of a graph and mark the first vertex in \( G \).
Last Time: Turing Machines

Intuitive notion of algorithm $\equiv$ Turing Machine
Solvable problem $\equiv$ Turing-decidable

$$
\mathcal{L} = \{\langle G \rangle \mid G \text{ is connected}\}
$$

$$
\langle G \rangle = 2; 1; 3; 4 \ # 1,2; 2,3; 1,3; 3,4
$$

($$\langle G \rangle$$ is the encoding of graph $G$ as a string.)

$M =$ Turing Machine that solves graph connectivity

**input:** $\langle G \rangle$, the encoding of a graph $G$.

1: Check that $\langle G \rangle$ is a valid encoding of a graph and mark the first vertex in $G$.

2: **REPEAT:** Find an edge in $G$ between a marked and an unmarked vertex.
   
   Mark the unmarked node or **GOTO** step 3 if there is no such edge.
Last Time: Turing Machines

Intuitive notion of algorithm \equiv \text{Turing Machine}
Solvable problem \equiv \text{Turing-decidable}

\[
\mathcal{L} = \{ \langle G \rangle \mid G \text{ is connected} \}
\]
\[
\langle G \rangle = 2; 1; 3; 4 \# 1,2; 2,3; 1,3; 3,4
\]

(\langle G \rangle \text{ is the encoding of graph } G \text{ as a string.})

\[
M = \text{Turing Machine that solves graph connectivity}
\]

\text{input: } \langle G \rangle, \text{ the encoding of a graph } G.
1: Check that \langle G \rangle \text{ is a valid encoding of a graph and mark the first vertex in } G.
2: \text{REPEAT: Find an edge in } G \text{ between a marked and an unmarked vertex.}
\hspace{0.5cm} \text{Mark the unmarked node or GOTO step 3 if there is no such edge.}
Last Time: Turing Machines

Intuitive notion of algorithm $\equiv$ Turing Machine
Solvable problem $\equiv$ Turing-decidable

$\mathcal{L} = \{\langle G \rangle \mid G \text{ is connected}\}$

$\langle G \rangle = 2; 1; 3; 4 \# 1,2; 2,3; 1,3; 3,4$

($\langle G \rangle$ is the encoding of graph $G$ as a string.)

$M =$ Turing Machine that solves graph connectivity
input: $\langle G \rangle$, the encoding of a graph $G$.
1: Check that $\langle G \rangle$ is a valid encoding of a graph and mark the first vertex in $G$.
2: REPEAT: Find an edge in $G$ between a marked and an unmarked vertex.
   Mark the unmarked node or GOTO step 3 if there is no such edge.
Intuitive notion of algorithm \equiv \text{Turing Machine}

Solvable problem \equiv \text{Turing-decidable}

\[ \mathcal{L} = \{ \langle G \rangle \mid G \text{ is connected} \} \]

\[ \langle G \rangle = 2; 1; 3; 4 \; \# \; 1,2; 2,3; 1,3; 3,4 \]

(\langle G \rangle \text{ is the encoding of graph } G \text{ as a string.})

\[ M = \text{Turing Machine that solves graph connectivity} \]

\textbf{input:} \langle G \rangle, \text{ the encoding of a graph } G.

1: Check that \langle G \rangle \text{ is a valid encoding of a graph and mark the first vertex in } G.

2: \text{REPEAT:} \text{ Find an edge in } G \text{ between a marked and an unmarked vertex.}

\hspace{1cm} \text{Mark the unmarked node or GOTO step 3 if there is no such edge.}
Last Time: Turing Machines

Intuitive notion of algorithm ≡ Turing Machine
Solvable problem ≡ Turing-decidable

\[ \mathcal{L} = \{ \langle G \rangle \mid G \text{ is connected} \} \]

\[ \langle G \rangle = 2; 1; 3; 4 \quad \# \quad 1,2; 2,3; 1,3; 3,4 \]

(\(\langle G \rangle\) is the encoding of graph \(G\) as a string.)

\[ M = \text{Turing Machine that solves graph connectivity} \]
\[ \text{input: } \langle G \rangle, \text{ the encoding of a graph } G. \]
1: Check that \(\langle G \rangle\) is a valid encoding of a graph and mark the first vertex in \(G\).
2: REPEAT: Find an edge in \(G\) between a marked and an unmarked vertex.
   Mark the unmarked node or GOTO step 3 if there is no such edge.
3: REJECT if there is an unmarked vertex remaining in \(G\); otherwise ACCEPT.

To tell your friend on the other coast about this fancy Turing Machine \(M\), encode its description into the bit-string \(\langle M \rangle\) and send over the telegraph.

You want to solve a different problem? Build another Turing Machine!
Today: Unsolvable Problems

1. Programmable Turing Machines.

2. Examples of unsolvable problems.
   - Post’s Correspondence Problem (PCP)?
   - HalfSum?
   - Auto-Grade?
   - Ultimate-Debugger?

3. $L_{TM}$: The language recognized by a Universal Turing Machine.
   - $L_{TM}$ is undecidable – cannot be solved!

4. Auto-Grade and Ultimate-Debugger do not exist.

5. What about HalfSum?
A Turing Machine $M$ has a binary encoding $\langle M \rangle$. Its input $w$ is a binary string.
A Turing Machine $M$ has a binary encoding $\langle M \rangle$. Its input $w$ is a binary string.

$\langle M \rangle \# w$ can be the input to another Turing Machine $U_{\text{TM}}$.

\[
U_{\text{TM}}(\langle M \rangle \# w) = \begin{cases} 
\text{halt with } \text{ACCEPT} & \text{if } M(w) = \text{halt with } \text{ACCEPT}; \\
\text{halt with } \text{REJECT} & \text{if } M(w) = \text{halt with } \text{REJECT}; \\
\text{loop forever} & \text{if } M(w) = \text{loop forever}; 
\end{cases}
\]

$U_{\text{TM}}$ outputs on $\langle M \rangle \# w$ whatever $M$ outputs on $w$. $U_{\text{TM}}$ simulates $M$. 

---

Creator: Malik Magdon-Ismail

Unsolvable Problems: 4 / 13

PCP and HALFSUM
A Turing Machine $M$ has a binary encoding $\langle M \rangle$. Its input $w$ is a binary string. $\langle M \rangle \# w$ can be the input to another Turing Machine $U_{TM}$.

\[
U_{TM}(\langle M \rangle \# w) = \begin{cases} 
\text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT}; \\
\text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT}; \\
\text{loop forever} & \text{if } M(w) = \text{loop forever}; 
\end{cases}
\]

$U_{TM}$ outputs on $\langle M \rangle \# w$ whatever $M$ outputs on $w$. $U_{TM}$ simulates $M$. 

Creator: Malik Magdon-Ismail
Unsolvable Problems: 4 / 13
PCP and HALFSUM →
A Turing Machine $M$ has a binary encoding $\langle M \rangle$. Its input $w$ is a binary string.

$\langle M \rangle \# w$ can be the input to another Turing Machine $U_{\text{TM}}$.

$$U_{\text{TM}}(\langle M \rangle \# w) = \begin{cases} 
\text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT}; \\
\text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT}; \\
\text{loop forever} & \text{if } M(w) = \text{loop forever}; 
\end{cases}$$

$U_{\text{TM}}$ outputs on $\langle M \rangle \# w$ whatever $M$ outputs on $w$. $U_{\text{TM}}$ simulates $M$.
A Turing Machine \( M \) has a binary encoding \( \langle M \rangle \). Its input \( w \) is a binary string.

\( \langle M \rangle \# w \) can be the input to another Turing Machine \( U_{\text{TM}} \).

\[
U_{\text{TM}}(\langle M \rangle \# w) = \begin{cases} 
\text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT}; \\
\text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT}; \\
\text{loop forever} & \text{if } M(w) = \text{loop forever};
\end{cases}
\]

\( U_{\text{TM}} \) outputs on \( \langle M \rangle \# w \) whatever \( M \) outputs on \( w \). \( U_{\text{TM}} \) simulates \( M \).
Programmable Turing Machine: Universal Turing Machine

A Turing Machine $M$ has a binary encoding $\langle M \rangle$. Its input $w$ is a binary string.

$\langle M \rangle #w$ can be the input to another Turing Machine $U_{\text{TM}}$.

\[
U_{\text{TM}}(\langle M \rangle #w) = \begin{cases} 
\text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT}; \\
\text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT}; \\
\text{loop forever} & \text{if } M(w) = \text{loop forever};
\end{cases}
\]

$U_{\text{TM}}$ outputs on $\langle M \rangle #w$ whatever $M$ outputs on $w$. $U_{\text{TM}}$ simulates $M$

**Challenge:** $U_{\text{TM}}$ is fixed but can simulate any $M$, even one with a million states.
Programmable Turing Machine: Universal Turing Machine

A Turing Machine $M$ has a binary encoding $\langle M \rangle$. Its input $w$ is a binary string.

$\langle M \rangle \# w$ can be the input to another Turing Machine $U_{\text{TM}}$.

\[
U_{\text{TM}}(\langle M \rangle \# w) = \begin{cases} 
\text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT}; \\
\text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT}; \\
\text{loop forever} & \text{if } M(w) = \text{loop forever}; 
\end{cases}
\]

$U_{\text{TM}}$ outputs on $\langle M \rangle \# w$ whatever $M$ outputs on $w$. $U_{\text{TM}}$ simulates $M$.

**Challenge:** $U_{\text{TM}}$ is fixed but can simulate any $M$, even one with a million states.

Entire simulation is done on the tape.
Programmable Turing Machine: Universal Turing Machine

A Turing Machine $M$ has a binary encoding $\langle M \rangle$. Its input $w$ is a binary string.

$\langle M \rangle \# w$ can be the input to another Turing Machine $U_{\text{TM}}$.

$$U_{\text{TM}}(\langle M \rangle \# w) = \begin{cases} \text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT}; \\ \text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT}; \\ \text{loop forever} & \text{if } M(w) = \text{loop forever}; \end{cases}$$

$U_{\text{TM}}$ outputs on $\langle M \rangle \# w$ whatever $M$ outputs on $w$. $U_{\text{TM}}$ simulates $M$.

**Challenge**: $U_{\text{TM}}$ is fixed but can simulate any $M$, even one with a million states.

Entire simulation is done on the tape.
A Turing Machine $M$ has a binary encoding $\langle M \rangle$. Its input $w$ is a binary string. $\langle M \rangle \# w$ can be the input to another Turing Machine $U_{\text{TM}}$.

$$U_{\text{TM}}(\langle M \rangle \# w) = \begin{cases} \text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT}; \\ \text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT}; \\ \text{loop forever} & \text{if } M(w) = \text{loop forever}; \end{cases}$$

$U_{\text{TM}}$ outputs on $\langle M \rangle \# w$ whatever $M$ outputs on $w$. $U_{\text{TM}}$ simulates $M$.

**Challenge:** $U_{\text{TM}}$ is fixed but can simulate any $M$, even one with a million states.

Entire simulation is done on the tape.
Programmable Turing Machine: Universal Turing Machine

A Turing Machine $M$ has a binary encoding $\langle M \rangle$. Its input $w$ is a binary string.

$\langle M \rangle # w$ can be the input to another Turing Machine $U_{TM}$.

$$U_{TM}(\langle M \rangle # w) = \begin{cases} 
\text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT}; \\
\text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT}; \\
\text{loop forever} & \text{if } M(w) = \text{loop forever};
\end{cases}$$

$U_{TM}$ outputs on $\langle M \rangle # w$ whatever $M$ outputs on $w$. $U_{TM}$ simulates $M$.

**Challenge:** $U_{TM}$ is fixed but can simulate any $M$, even one with a million states.

Entire simulation is done on the tape.
Programmable Turing Machine: Universal Turing Machine

A Turing Machine $M$ has a binary encoding $\langle M \rangle$. Its input $w$ is a binary string.

$\langle M \rangle \# w$ can be the input to another Turing Machine $U_{\text{TM}}$.

$$ U_{\text{TM}}(\langle M \rangle \# w) = \begin{cases} 
\text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT}; \\
\text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT}; \\
\text{loop forever} & \text{if } M(w) = \text{loop forever}; 
\end{cases} $$

$U_{\text{TM}}$ outputs on $\langle M \rangle \# w$ whatever $M$ outputs on $w$. $U_{\text{TM}}$ simulates $M$.

**Challenge:** $U_{\text{TM}}$ is fixed but can simulate any $M$, even one with a million states.

Entire simulation is done on the tape.
Post’s Correspondence Problem (PCP) and **HALFSUM**

**PCP:** Consider 3 dominos:

<table>
<thead>
<tr>
<th></th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>00</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>
Post’s Correspondence Problem (PCP) and \textbf{HALFSUM}

\textbf{PCP:} Consider 3 dominos: 
\begin{align*}
    d_1 & = \begin{array}{c}
        0 \\
        100
    \end{array} & 
    d_2 & = \begin{array}{c}
        01 \\
        00
    \end{array} & 
    d_3 & = \begin{array}{c}
        110 \\
        11
    \end{array}
\end{align*}

\[
    d_3d_2d_3d_1 = \begin{array}{cccc}
        110 & 01 & 110 & 0 \\
        11 & 00 & 11 & 100
    \end{array} = \begin{array}{c}
        110011100 \\
        110011100
    \end{array}
\]

\begin{itemize}
    \item Top and bottom strings match. That’s the goal.
\end{itemize}
Post’s Correspondence Problem (PCP) and HALFSUM

**PCP:** Consider 3 dominos: 
\[
\begin{align*}
\begin{array}{c|c|c}
& d_1 & d_2 & d_3 \\
\hline
0 & 01 & 110 \\
\hline
100 & 00 & 11 \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
d_3d_2d_3d_1 &= \begin{array}{c|c|c|c}
110 & 01 & 110 & 0 \\
\hline
11 & 00 & 11 & 100 \\
\end{array} = \begin{array}{c|c|c|c}
110011100 & 110011100 \\
\end{array}
\end{align*}
\]

Top and bottom strings match. That’s the goal.

**INPUT:** Dominos \(\{d_1, d_2, \ldots, d_n\}\). For example \(\left\{ \begin{array}{c|c|c}
10 & 011 & 101 \\
\hline
101 & 11 & 011 \\
\end{array} \right\} \).

**TASK:** Can one line up finitely many dominos so that the top and bottom strings match?
**PCP:** Consider 3 dominos: \[ \begin{array}{ccc}
  d_1 & d_2 & d_3 \\
  0 & 01 & 110 \\
  100 & 00 & 11 \\
\end{array} \]

\[d_3 d_2 d_3 d_1 = \begin{array}{cccc}
  110 & 01 & 110 & 0 \\
  11 & 00 & 11 & 100 \\
\end{array} = \begin{array}{c}
  110011100 \\
  110111000 \\
\end{array}\]

Top and bottom strings match. That’s the goal.

**INPUT:** Dominos \( \{d_1, d_2, \ldots, d_n\} \). For example \( \{1010, 0111, 1011\} \).

**TASK:** Can one line up finitely many dominos so that the top and bottom strings match?

**HalfSum:** Consider the multiset \( S = \{1, 1, 1, 3, 4, 4, 5, 6, 9\} \), and subset \( A = \{1, 3, 4, 9\} \).

\[\text{sum}(A) = 17 = \frac{1}{2} \times \text{sum}(S).\]
Post’s Correspondence Problem (PCP) and **HALFSUM**

**PCP:** Consider 3 dominos:

\[
\begin{array}{c}
\text{d}_1 \\
0 \\
100
\end{array}
\begin{array}{c}
\text{d}_2 \\
01 \\
00
\end{array}
\begin{array}{c}
\text{d}_3 \\
110 \\
11
\end{array}
\]

\[d_3d_2d_3d_1 = \begin{array}{ccc}
110 & 01 & 110 \\
11 & 00 & 11
\end{array} = \begin{array}{c}
11001100 \\
11001100
\end{array}\]

← Top and bottom strings match. That’s the goal.

**INPUT:** Dominos \{d_1, d_2, \ldots, d_n\}. For example \{\begin{array}{c}
10101 \\
01111 \\
101011
\end{array}\}.

**TASK:** Can one line up finitely many dominos so that the top and bottom strings match?

**HALFSUM:** Consider the multiset \(S = \{1, 1, 1, 3, 4, 4, 5, 6, 9\}\), and subset \(A = \{1, 3, 4, 9\}\).

\[
\text{sum}(A) = 17 = \frac{1}{2} \times \text{sum}(S).
\]

**INPUT:** Multiset \(S = \{x_1, x_2, \ldots, x_n\}\). For example, \(S = \{1, 1, 1, 3, 4, 4, 5, 6, 9\}\).

**TASK:** Is there a subset whose sum is \(\frac{1}{2} \times \text{sum}(S) = \frac{1}{2} \times (x_1 + x_2 + \cdots + x_n)\)?

Creator: Malik Magdon-Ismail
Unsolvable Problems: 5 / 13
Your first CS assignment: Write a program to print “Hello World!” and halt.

CS1: 700+ submissions!

Naturally, we do not grade these by hand.

Auto-Grade: runs each submission and determines if its correct.
Your first CS assignment: Write a program to print “Hello World!” and halt.

CS1: 700+ submissions!

Naturally, we do not grade these by hand.

Auto-Grade: runs each submission and determines if its correct.

What does Auto-Grade say for this program:

```c
n = 4;
while(n > 0){
    if(n is not a sum of two primes){
        print("Hello World!") and exit;
    }
    n ← n + 2;
}
```
Wouldn’t it be nice to have the **ULTIMATE-DEBUGGER**. \(\leftarrow\) solves the *Halting Problem*

\[
\begin{align*}
\text{HALTS} & = \left\{ \begin{array}{ll}
\text{YES} & \text{if program halts} \\
\text{NO} & \text{if program infinitely loops}
\end{array} \right.
\end{align*}
\]
Wouldn’t it be nice to have the **Ultimate-Debugger**.

\[ \text{Halts} \begin{cases} n = 4; \\
\text{while}(n > 0)\{ \\
\text{if}(n \text{ is not a sum of two primes})\{ \\
\quad \text{print("Hello World!") and exit;} \\
\} \\
\quad n \leftarrow n + 2; \\
\} \end{cases} = \begin{cases} \text{YES} & \text{if program halts} \\
\text{NO} & \text{if program infinitely loops} \end{cases} \]

- We can grade the students program correctly.
- We can solve Goldbach’s conjecture.
- Just think what you could do with **Ultimate-Debugger**.
  - No more infinite looping programs.
Verification: Does A Program Successfully Terminate?
Verification: Does A Program Successfully Terminate?

\[ \mathcal{L}_{TM} = \{ \langle M \rangle \# w \mid M \text{ is a Turing Machine and } M \text{ accepts } w \} . \]
Verification: Does A Program Successfully Terminate?

\[ \mathcal{L}_{\text{TM}} = \{ \langle M \rangle \# w \mid M \text{ is a Turing Machine and } M \text{ accepts } w \} \].

\( U_{\text{TM}} \) is a recognizer for \( \mathcal{L}_{\text{TM}} \).
Verification: Does A Program Successfully Terminate?

\[ \mathcal{L}_{\text{TM}} = \{ \langle M \rangle \# w \mid M \text{ is a Turing Machine and } M \text{ accepts } w \} \].

\( U_{\text{TM}} \) is a recognizer for \( \mathcal{L}_{\text{TM}} \).

Is there a Turing Machine \( A_{\text{TM}} \) which decides \( \mathcal{L}_{\text{TM}} \)?

- A decider must always halt with an answer.
- \( U_{\text{TM}} \) may loop forever if \( M \) loops forever on \( w \).
Verification: Does A Program Successfully Terminate?

\[ \mathcal{L}_{\text{TM}} = \{ \langle M \rangle \# w \mid M \text{ is a Turing Machine and } M \text{ accepts } w \} . \]

\( U_{\text{TM}} \) is a recognizer for \( \mathcal{L}_{\text{TM}} \).

Is there a Turing Machine \( A_{\text{TM}} \) which decides \( \mathcal{L}_{\text{TM}} \)?

- A decider must always halt with an answer.
- \( U_{\text{TM}} \) may loop forever if \( M \) loops forever on \( w \).
- Question: What do these mean: \( M(\langle M \rangle) \) and \( A_{\text{TM}}(\langle M \rangle \# \langle M \rangle) \)?
$L_{\text{TM}} = \{ \langle M \rangle \# w \mid M \text{ is a Turing Machine and } M \text{ accepts } w \}$. 

$U_{\text{TM}}$ is a recognizer for $L_{\text{TM}}$.

Is there a Turing Machine $A_{\text{TM}}$ which decides $L_{\text{TM}}$?

- A decider must always halt with an answer.
- $U_{\text{TM}}$ may loop forever if $M$ loops forever on $w$.
- Question: What do these mean: $M(\langle M \rangle)$ and $A_{\text{TM}}(\langle M \rangle \# \langle M \rangle)$?

A diabolical Turing Machine $D$ built from $A_{\text{TM}}$:

\begin{quote}
$D$ = “Diagonal” Turing Machine derived from $A_{\text{TM}}$ (the decider for $L_{\text{TM}}$) 

\textbf{input:} $\langle M \rangle$ where $M$ is a Turing Machine.
\end{quote}
Verification: Does A Program Successfully Terminate?

\[ \mathcal{L}_{\text{TM}} = \{ \langle M \rangle \# w | M \text{ is a Turing Machine and } M \text{ accepts } w \}. \]

\( U_{\text{TM}} \) is a recognizer for \( \mathcal{L}_{\text{TM}} \).

Is there a Turing Machine \( A_{\text{TM}} \) which decides \( \mathcal{L}_{\text{TM}} \)?

- A decider must *always* halt with an answer.
- \( U_{\text{TM}} \) may loop forever if \( M \) loops forever on \( w \).
- Question: What do these mean: \( M(\langle M \rangle) \) and \( A_{\text{TM}}(\langle M \rangle \# \langle M \rangle) \) ?

A diabolical Turing Machine \( D \) built from \( A_{\text{TM}} \):

\[
D = \text{“Diagonal” Turing Machine derived from } A_{\text{TM}} \text{ (the decider for } \mathcal{L}_{\text{TM}}) \\
\text{input: } \langle M \rangle \text{ where } M \text{ is a Turing Machine.} \\
1: \text{Run } A_{\text{TM}} \text{ with input } \langle M \rangle \# \langle M \rangle.
\]

\( D \) does the opposite of \( A_{\text{TM}} \). Is \( D \) a decider?
**Verification: Does A Program Successfully Terminate?**

\[ \mathcal{L}_{TM} = \{ \langle M \rangle \# w \mid M \text{ is a Turing Machine and } M \text{ accepts } w \} \].

\( U_{TM} \) is a recognizer for \( \mathcal{L}_{TM} \).

Is there a Turing Machine \( A_{TM} \) which decides \( \mathcal{L}_{TM} \)?

- A decider must *always* halt with an answer.
- \( U_{TM} \) may loop forever if \( M \) loops forever on \( w \).
- Question: What do these mean: \( M(\langle M \rangle) \) and \( A_{TM}(\langle M \rangle \# \langle M \rangle) \) ?

A diabolical Turing Machine \( D \) built from \( A_{TM} \):

\[
D = \text{“Diagonal” Turing Machine derived from } A_{TM} (\text{the decider for } \mathcal{L}_{TM})
\]

**input:** \( \langle M \rangle \) where \( M \) is a Turing Machine.

1: Run \( A_{TM} \) with input \( \langle M \rangle \# \langle M \rangle \).
2: If \( A_{TM} \) accepts then REJECT; otherwise (\( A_{TM} \) rejects) ACCEPT

\( D \) does the *opposite* of \( A_{TM} \). Is \( D \) a decider?
**Theorem.** \( A_{\text{TM}} \) does not exist (\( \mathcal{L}_{\text{TM}} \) Cannot be Solved)

\[ A_{\text{TM}} \text{ exists } \rightarrow D \text{ exists.} \]

\( D \) exists means it will appear on the list of all Turing Machines,

\[ \langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D \rangle, \ldots \]
**Theorem.** \( A_{TM} \) does not exist \((L_{TM} \text{ Cannot be Solved})\)

\( A_{TM} \) exists \(\rightarrow D \) exists.

\( D \) exists means it will appear on the list of all Turing Machines,
\[
\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D \rangle, \ldots
\]

Consider what happens when \( M_i \) runs on \( \langle M_j \rangle \), that is \( A_{TM}(\langle M_i \rangle \# \langle M_j \rangle) \).

<table>
<thead>
<tr>
<th>( A_{TM}(\langle M_i \rangle # \langle M_j \rangle) )</th>
<th>( \langle M_1 \rangle )</th>
<th>( \langle M_2 \rangle )</th>
<th>( \langle M_3 \rangle )</th>
<th>( \langle M_4 \rangle )</th>
<th>( \langle D \rangle )</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle M_1 \rangle )</td>
<td>( \langle M_1 \rangle )</td>
<td>( \langle M_2 \rangle )</td>
<td>( \langle M_3 \rangle )</td>
<td>( \langle M_4 \rangle )</td>
<td>( \langle D \rangle )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( \langle M_2 \rangle )</td>
<td>( \langle M_2 \rangle )</td>
<td>( \langle M_3 \rangle )</td>
<td>( \langle M_4 \rangle )</td>
<td>( \langle D \rangle )</td>
<td>( \ldots )</td>
<td></td>
</tr>
<tr>
<td>( \langle M_3 \rangle )</td>
<td>( \langle M_3 \rangle )</td>
<td>( \langle M_4 \rangle )</td>
<td>( \langle D \rangle )</td>
<td>( \ldots )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \langle M_4 \rangle )</td>
<td>( \langle M_4 \rangle )</td>
<td>( \langle D \rangle )</td>
<td>( \ldots )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \langle D \rangle )</td>
<td>( \ldots )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Theorem. $A_{TM}$ does not exist ($\mathcal{L}_{TM}$ Cannot be Solved)

$A_{TM}$ exists $\rightarrow$ $D$ exists.

$D$ exists means it will appear on the list of all Turing Machines,

$$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D \rangle, \ldots$$

Consider what happens when $M_i$ runs on $\langle M_j \rangle$, that is $A_{TM}(\langle M_i \rangle \# \langle M_j \rangle)$.

<table>
<thead>
<tr>
<th>$A_{TM}(\langle M_i \rangle # \langle M_j \rangle)$</th>
<th>$\langle M_1 \rangle$</th>
<th>$\langle M_2 \rangle$</th>
<th>$\langle M_3 \rangle$</th>
<th>$\langle M_4 \rangle$</th>
<th>$\langle D \rangle$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle M_1 \rangle$</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_2 \rangle$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\langle M_3 \rangle$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\langle M_4 \rangle$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\langle D \rangle$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\vdots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Theorem.** $A_{\text{TM}}$ does not exist ($L_{\text{TM}}$ Cannot be Solved)

$A_{\text{TM}}$ exists $\rightarrow D$ exists.

$D$ exists means it will appear on the list of all Turing Machines, 
\[\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D \rangle, \ldots\]

Consider what happens when $M_i$ runs on $\langle M_j \rangle$, that is $A_{\text{TM}}(\langle M_i \rangle \# \langle M_j \rangle)$.

\[
\begin{array}{cccccc}
A_{\text{TM}}(\langle M_i \rangle \# \langle M_j \rangle) & \langle M_1 \rangle & \langle M_2 \rangle & \langle M_3 \rangle & \langle M_4 \rangle & \langle D \rangle & \cdots \\
\langle M_1 \rangle & \text{ACCEPT} & \text{ACCEPT} & \text{REJECT} & \text{ACCEPT} & \text{ACCEPT} & \cdots \\
\langle M_2 \rangle \\
\langle M_3 \rangle \\
\langle M_4 \rangle \\
\langle D \rangle & \text{REJECT} \\
\vdots & \\
\end{array}
\]

$D(\langle M_i \rangle)$ does the opposite of $A_{\text{TM}}(\langle M_i \rangle \# \langle M_i \rangle)$. 
Theorem. \( A_{TM} \) does not exist (\( L_{TM} \) Cannot be Solved)

\( A_{TM} \) exists \( \rightarrow D \) exists.

\( D \) exists means it will appear on the list of all Turing Machines,

\[ \langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D \rangle, \ldots \]

Consider what happens when \( M_i \) runs on \( \langle M_j \rangle \), that is \( A_{TM}(\langle M_i \rangle \# \langle M_j \rangle) \).

<table>
<thead>
<tr>
<th>( A_{TM}(\langle M_i \rangle # \langle M_j \rangle) )</th>
<th>( \langle M_1 \rangle )</th>
<th>( \langle M_2 \rangle )</th>
<th>( \langle M_3 \rangle )</th>
<th>( \langle M_4 \rangle )</th>
<th>( \langle D \rangle )</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle M_1 \rangle )</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( \langle M_2 \rangle )</td>
<td>REJECT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( \langle M_3 \rangle )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \langle M_4 \rangle )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \langle D \rangle )</td>
<td>REJECT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \vdots )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( D(\langle M_i \rangle) \) does the opposite of \( A_{TM}(\langle M_i \rangle \# \langle M_i \rangle) \).
**Theorem.** \( A_{TM} \) does not exist (\( L_{TM} \) Cannot be Solved)

\( A_{TM} \) exists \( \rightarrow D \) exists.

\( D \) exists means it will appear on the list of all Turing Machines,
\[ \langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D \rangle, \ldots \]

Consider what happens when \( M_i \) runs on \( \langle M_j \rangle \), that is \( A_{TM}(\langle M_i \rangle \# \langle M_j \rangle) \).

<table>
<thead>
<tr>
<th>( A_{TM}(\langle M_i \rangle # \langle M_j \rangle) )</th>
<th>( \langle M_1 \rangle )</th>
<th>( \langle M_2 \rangle )</th>
<th>( \langle M_3 \rangle )</th>
<th>( \langle M_4 \rangle )</th>
<th>( \langle D \rangle )</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle M_1 \rangle )</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( \langle M_2 \rangle )</td>
<td>REJECT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( \langle M_3 \rangle )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \langle M_4 \rangle )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \langle D \rangle )</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \vdots )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( D(\langle M_i \rangle) \) does the *opposite* of \( A_{TM}(\langle M_i \rangle \# \langle M_i \rangle) \).
Theorem. $A_{TM}$ does not exist ($\mathcal{L}_{TM}$ Cannot be Solved)

$A_{TM}$ exists $\rightarrow$ $D$ exists.

$D$ exists means it will appear on the list of all Turing Machines,

$$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D \rangle, \ldots$$

Consider what happens when $M_i$ runs on $\langle M_j \rangle$, that is $A_{TM}(\langle M_i \rangle \# \langle M_j \rangle)$.

<table>
<thead>
<tr>
<th>$A_{TM}(\langle M_i \rangle # \langle M_j \rangle)$</th>
<th>$\langle M_1 \rangle$</th>
<th>$\langle M_2 \rangle$</th>
<th>$\langle M_3 \rangle$</th>
<th>$\langle M_4 \rangle$</th>
<th>$\langle D \rangle$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle M_1 \rangle$</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_2 \rangle$</td>
<td>REJECT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_3 \rangle$</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_4 \rangle$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\langle D \rangle$</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\vdots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$D(\langle M_i \rangle)$ does the opposite of $A_{TM}(\langle M_i \rangle \# \langle M_i \rangle)$.
Theorem. $A_{TM}$ does not exist ($L_{TM}$ Cannot be Solved)

$A_{TM}$ exists $\rightarrow D$ exists.

$D$ exists means it will appear on the list of all Turing Machines,

$$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D \rangle, \ldots$$

Consider what happens when $M_i$ runs on $\langle M_j \rangle$, that is $A_{TM}(\langle M_i \rangle \# \langle M_j \rangle)$.

<table>
<thead>
<tr>
<th>$A_{TM}(\langle M_i \rangle # \langle M_j \rangle)$</th>
<th>$\langle M_1 \rangle$</th>
<th>$\langle M_2 \rangle$</th>
<th>$\langle M_3 \rangle$</th>
<th>$\langle M_4 \rangle$</th>
<th>$\langle D \rangle$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle M_1 \rangle$</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_2 \rangle$</td>
<td>REJECT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_3 \rangle$</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_4 \rangle$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\langle D \rangle$</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\vdots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$D(\langle M_i \rangle)$ does the opposite of $A_{TM}(\langle M_i \rangle \# \langle M_i \rangle)$.
Theorem. $A_{TM}$ does not exist ($\mathcal{L}_{TM}$ Cannot be Solved)

$A_{TM}$ exists $\rightarrow D$ exists.

$D$ exists means it will appear on the list of all Turing Machines,
$$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D \rangle, \ldots$$

Consider what happens when $M_i$ runs on $\langle M_j \rangle$, that is $A_{TM}(\langle M_i \rangle \# \langle M_j \rangle)$.

<table>
<thead>
<tr>
<th>$A_{TM}(\langle M_i \rangle # \langle M_j \rangle)$</th>
<th>$\langle M_1 \rangle$</th>
<th>$\langle M_2 \rangle$</th>
<th>$\langle M_3 \rangle$</th>
<th>$\langle M_4 \rangle$</th>
<th>$\langle D \rangle$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle M_1 \rangle$</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_2 \rangle$</td>
<td>REJECT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_3 \rangle$</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_4 \rangle$</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle D \rangle$</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\vdots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$D(\langle M_i \rangle)$ does the opposite of $A_{TM}(\langle M_i \rangle \# \langle M_i \rangle)$.
**Theorem.** \( A_{TM} \) does not exist (\( L_{TM} \) Cannot be Solved)

\( A_{TM} \) exists \( \rightarrow \) \( D \) exists.

\( D \) exists means it will appear on the list of all Turing Machines,
\[
\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D \rangle, \ldots
\]

Consider what happens when \( M_i \) runs on \( \langle M_j \rangle \), that is \( A_{TM}(\langle M_i \rangle \# \langle M_j \rangle) \).

\[
\begin{array}{c|cccccc}
A_{TM}(\langle M_i \rangle \# \langle M_j \rangle) & \langle M_1 \rangle & \langle M_2 \rangle & \langle M_3 \rangle & \langle M_4 \rangle & \langle D \rangle & \ldots \\
\hline
\langle M_1 \rangle & \text{ACCEPT} & \text{ACCEPT} & \text{REJECT} & \text{ACCEPT} & \text{ACCEPT} & \ldots \\
\langle M_2 \rangle & \text{REJECT} & \text{REJECT} & \text{REJECT} & \text{ACCEPT} & \text{ACCEPT} & \ldots \\
\langle M_3 \rangle & \text{ACCEPT} & \text{ACCEPT} & \text{REJECT} & \text{REJECT} & \text{ACCEPT} & \ldots \\
\langle M_4 \rangle & \text{ACCEPT} & \text{REJECT} & \text{REJECT} & \text{REJECT} & \text{ACCEPT} & \ldots \\
\langle D \rangle & \text{REJECT} & \text{ACCEPT} & \text{ACCEPT} & \text{ACCEPT} & \ldots \\
\vdots
\end{array}
\]

\( D(\langle M_i \rangle) \) does the opposite of \( A_{TM}(\langle M_i \rangle \# \langle M_i \rangle) \).
Theorem. $A_{TM}$ does not exist ($L_{TM}$ Cannot be Solved)

$A_{TM}$ exists $\rightarrow D$ exists.

$D$ exists means it will appear on the list of all Turing Machines, 

$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D \rangle, \ldots$

Consider what happens when $M_i$ runs on $\langle M_j \rangle$, that is $A_{TM}(\langle M_i \rangle\#\langle M_j \rangle)$.

<table>
<thead>
<tr>
<th>$A_{TM}(\langle M_i \rangle#\langle M_j \rangle)$</th>
<th>$\langle M_1 \rangle$</th>
<th>$\langle M_2 \rangle$</th>
<th>$\langle M_3 \rangle$</th>
<th>$\langle M_4 \rangle$</th>
<th>$\langle D \rangle$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle M_1 \rangle$</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_2 \rangle$</td>
<td>REJECT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_3 \rangle$</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_4 \rangle$</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle D \rangle$</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>

$D(\langle M_i \rangle)$ does the opposite of $A_{TM}(\langle M_i \rangle\#\langle M_i \rangle)$.
Theorem. $A_{TM}$ does not exist ($L_{TM}$ Cannot be Solved)

$A_{TM}$ exists $\rightarrow D$ exists.

$D$ exists means it will appear on the list of all Turing Machines,

$$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D \rangle, \ldots$$

Consider what happens when $M_i$ runs on $\langle M_j \rangle$, that is $A_{TM}(\langle M_i \rangle \# \langle M_j \rangle)$.

| $A_{TM}(\langle M_i \rangle \# \langle M_j \rangle)$ | $\langle M_1 \rangle$ | $\langle M_2 \rangle$ | $\langle M_3 \rangle$ | $\langle M_4 \rangle$ | $\langle D \rangle$ | $\ldots$
|---|---|---|---|---|---|---|
| $\langle M_1 \rangle$ | **ACCEPT** | **ACCEPT** | REJECT | ACCEPT | ACCEPT | $\ldots$
| $\langle M_2 \rangle$ | REJECT | **REJECT** | REJECT | ACCEPT | ACCEPT | $\ldots$
| $\langle M_3 \rangle$ | ACCEPT | ACCEPT | **REJECT** | REJECT | ACCEPT | $\ldots$
| $\langle M_4 \rangle$ | ACCEPT | REJECT | REJECT | **REJECT** | ACCEPT | $\ldots$
| $\langle D \rangle$ | REJECT | ACCEPT | ACCEPT | ACCEPT | **REJECT?** | $\ldots$
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$

$D(\langle M_i \rangle)$ does the *opposite* of $A_{TM}(\langle M_i \rangle \# \langle M_i \rangle)$.
**Theorem.** \( A_{TM} \) does not exist (\( L_{TM} \) Cannot be Solved)

\( A_{TM} \) exists \( \rightarrow \) \( D \) exists.

\( D \) exists means it will appear on the list of all Turing Machines,

\[ \langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D \rangle, \ldots \]

Consider what happens when \( M_i \) runs on \( \langle M_j \rangle \), that is \( A_{TM}(\langle M_i \rangle \# \langle M_j \rangle) \).

<table>
<thead>
<tr>
<th>( A_{TM}(\langle M_i \rangle # \langle M_j \rangle) )</th>
<th>( \langle M_1 \rangle )</th>
<th>( \langle M_2 \rangle )</th>
<th>( \langle M_3 \rangle )</th>
<th>( \langle M_4 \rangle )</th>
<th>( \langle D \rangle )</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle M_1 \rangle )</td>
<td><strong>ACCEPT</strong></td>
<td><strong>ACCEPT</strong></td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( \langle M_2 \rangle )</td>
<td>REJECT</td>
<td><strong>REJECT</strong></td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( \langle M_3 \rangle )</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td><strong>REJECT</strong></td>
<td><strong>REJECT</strong></td>
<td>ACCEPT</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( \langle M_4 \rangle )</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td><strong>REJECT</strong></td>
<td><strong>REJECT</strong></td>
<td>ACCEPT</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( \langle D \rangle )</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>ACCEPT?</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \ldots )</td>
</tr>
</tbody>
</table>

\( D(\langle M_i \rangle) \) does the **opposite** of \( A_{TM}(\langle M_i \rangle \# \langle M_i \rangle) \).
No *general* program/algorithm to analyze *any* other program $M$ and tell if $M$ will accept or not a particular input. 🙁
Suppose \texttt{Ultimate-Debugger} $H_{\text{TM}}$ exists and \textit{decides} if any other program halts.

We can use $H_{\text{TM}}$ to construct a solver $A_{\text{TM}}$ for $L_{\text{TM}}$. 

No \textit{general} program/algorithm to analyze \textit{any} other program $M$ and tell if $M$ will accept or not a particular input. 😞

No \texttt{Ultimate-Debugger} to analyze other programs and tell if they halt. 😞
No *general* program/algorithm to analyze *any* other program $M$ and tell if $M$ will accept or not a particular input.

Suppose **Ultimate-Debugger** $H_{TM}$ exists and *decides* if any other program halts.

We can use $H_{TM}$ to construct a solver $A_{TM}$ for $L_{TM}$.

$$A_{TM} = \text{Turing Machine derived from } H_{TM} \text{ (the decider for } L_{HALT})$$

**input:** $\langle M \rangle \# w$ where $M$ is a Turing Machine and $w$ an input to $M$. 

No **Ultimate-Debugger** to analyze other programs and tell if they halt. 😞
Ultimate-Debugger and Auto-Grade Don’t Exist

No general program/algorithm to analyze any other program $M$ and tell if $M$ will accept or not a particular input.

Suppose Ultimate-Debugger $H_{TM}$ exists and decides if any other program halts.

We can use $H_{TM}$ to construct a solver $A_{TM}$ for $L_{TM}$.

\[
A_{TM} = \text{Turing Machine derived from } H_{TM} \text{ (the decider for } L_{HALT})
\]

**input:** $\langle M \rangle \# w$ where $M$ is a Turing Machine and $w$ an input to $M$.

1: Run $H_{TM}$ on input $\langle M \rangle \# w$. If $H_{TM}$ rejects, then REJECT.
2: Run $U_{TM}$ on input $\langle M \rangle \# w$ and output the decision $U_{TM}$ gives.
No *general* program/algorithm to analyze *any* other program $M$ and tell if $M$ will accept or not a particular input.

Suppose **Ultimate-Debugger** $H_{\text{TM}}$ exists and *decides* if any other program halts.

We can use $H_{\text{TM}}$ to construct a solver $A_{\text{TM}}$ for $L_{\text{TM}}$.

$$A_{\text{TM}} = \text{Turing Machine derived from } H_{\text{TM}} \text{ (the decider for } L_{\text{HALT}})$$

- **input**: $\langle M \rangle \# w$ where $M$ is a Turing Machine and $w$ an input to $M$.
- 1: Run $H_{\text{TM}}$ on input $\langle M \rangle \# w$. If $H_{\text{TM}}$ rejects, then REJECT.
- 2: Run $U_{\text{TM}}$ on input $\langle M \rangle \# w$ and output the decision $U_{\text{TM}}$ gives.

**Exercise.** Show that **Auto-Grade** does not exist.

**Exercise.** Show that **HalfSum** is solvable by giving a decider.
No general program/algorithm to analyze any other program $M$ and tell if $M$ will accept or not a particular input. 😞

No ULTIMATE-Debugger to analyze other programs and tell if they halt. 😞

No AUTO-GRADE for CS-1 programs. 😞

No solver for PCP. 😞

Suppose ULTIMATE-Debugger $H_{TM}$ exists and decides if any other program halts.

We can use $H_{TM}$ to construct a solver $A_{TM}$ for $L_{TM}$.

$$A_{TM} = \text{Turing Machine derived from } H_{TM} \text{ (the decider for } L_{HALT})$$

**input:** $⟨M⟩#w$ where $M$ is a Turing Machine and $w$ an input to $M$.

1: Run $H_{TM}$ on input $⟨M⟩#w$. If $H_{TM}$ rejects, then REJECT.

2: Run $U_{TM}$ on input $⟨M⟩#w$ and output the decision $U_{TM}$ gives.

**Exercise.** Show that AUTO-GRADE does not exist.

**Exercise.** Show that HALFSUM is solvable by giving a decider.
DFA
(no external memory)
(regular expressions)
\{0^*1^*, 0^{3n+1}\}
The Landscape

DFA
(no external memory)
(regular expressions)
\{0^*1^*, 0^{3n+1}\}

CFG
(stack)
\{0^n1^n\},
\{ww^R\}
The Landscape

DFA
(no external memory)
(regular expressions)
\{ *01* \}, \{ 0^{3n+1} \}

CFG
(stack)
\{ 0^n1^n \},
\{ \text{ww} \}, \{ 0^{2n} \},
\{ 0^n1^n0^n \}

TM-Decider
(RAM)
\{ \text{ww} \}, \{ 0^{2n} \},
\{ 0^n1^n0^n \}

HALFSUM
The Landscape

DFA  
(no external memory)  
(regular expressions)  
\{\ast 01\ast\}, \{0\cdot 3^{n+1}\}

CFG  
(stack)  
\{0^n1^n\},  
\{ww^R\}

TM-Decider  
(RAM)  
\{ww\}, \{0^{2^n}\},  
\{0^n1^n0^n\}  
HalfSum

TM-Recognizer  
\mathcal{L}_{TM}

Ultimate-Debugger
Auto-Grade
PCP
The Landscape

DFA (no external memory) (regular expressions) 
\{ \ast 01 \ast \}, \{ 0 \ast 3n+1 \}

CFG (stack) 
\{ 0^n1^n \}, 
\{ w w^R \}

TM-Decider (RAM) 
\{ w w \}, \{ 0^{2n} \}, 
\{ 0^n1^n0^n \}

TM-Recognizer 
\mathcal{L}_{\text{TM}}
ULTIMATE-DEBUGGER
AUTO-GRADE
PCP

Non-Recognizable 
\overline{\mathcal{L}_{\text{TM}}}, \overline{\mathcal{L}_{\text{HALT}}}
most languages

Creator: Malik Magdon-Ismail
Unsolvable Problems: 11 / 13
The Path Forward →
The Path Forward: Focus on Decidable Problems
The Path Forward: Focus on Decidable Problems

FOCS

Theory of Computing

Discrete Math
The Path Forward: Focus on Decidable Problems

FOCS

Decider
$U_{TM} = \text{computer}$
$TM = \text{Algorithm}$

Theory of Computing

CFG Parsing

DFA RegExp

Discrete Math

Creator: Malik Magdon-Ismail
Unsolvable Problems: 12 / 13
Epic Disasters →
The Path Forward: Focus on Decidable Problems

- Decider
  - $U_{TM} = \text{computer}$
  - $TM = \text{Algorithm}$

- CFG Parsing
- DFA RegExp

- FOCS
- Proof, logic
  - \textbf{INDUCTION}
- Recursion
  - Struct. Induction
- Sums, Asymptotics
- Number theory
- Graphs
- Counting
- Probability

Creator: Malik Magdon-Ismail
Unsolvable Problems: 12 / 13
The Path Forward: Focus on Decidable Problems

- Decider
  $U_{TM} = \text{computer}$
  $TM = \text{Algorithm}$

- CFG
  Parsing

- DFA
  RegExp

- Proof, logic
  INDUCTION

- Recursion
  Struct. Induction

- Sums, Asymptotics

- Number theory

- Graph theory

- Linear Algebra

- Probability Theory

- Multivariate Calc.
The Path Forward: Focus on Decidable Problems

Decider

$U_{tm} = \text{computer}$

$TM = \text{Algorithm}$

CFG

Parsing

DFA

RegExp

Proof, logic

INDUCTION

Recursion

Struct. Induction

Sums, Asymptotics

Number theory

Graphs

Counting

Probability

FAST (P)

Polynomial

FAST (NP)

Unbounded Parallelism

SLOW

Exponential

Boolean Circuits

Chapters 28 & 29

P = NP?

Unsolvable Problems: 12 / 13

Creator: Malik Magdon-Ismail

Epic Disasters
The Path Forward: Focus on Decidable Problems

Decider $U_{tm} = \text{computer}$
$TM = \text{Algorithm}$

$\text{CFG Parsing}$
$\text{DFA RegExp}$

$\text{Proof, logic}$
$\text{INDUCTION}$

$\text{Recursion}$
$\text{Struct. Induction}$

$\text{Sums, Asymptotics}$
$\text{Number theory}$

$\text{Graphs}$
$\text{Counting}$

$\text{Probability}$

$\text{FAST (P)}$ Polynomial
$\text{FAST (NP)}$ Unbounded Parallelism
$\text{SLOW Exponential}$
$\text{Boolean Circuits}$

$P = NP?$

$\text{Computability & Complexity}$
$\text{Algorithms & DS}$
- Approximation
- Randomized
- Distributed

$\text{Cryptography}$

$\text{Data}$
- ML/AI/DM/NLP
- Vision
- Graphics
- Comp. Finance

$\text{Networks}$
- Computers
- Social
- Data (e.g. www)

$\text{Robotics}$
$\text{Security}$

$\text{Programming Languages}$
- Compilers
- Distributed

$\text{Program Analysis}$
- Testing
- Verification

Creator: Malik Magdon-Ismail
The Path Forward: Focus on Decidable Problems

Decider
$U_{tm} = \text{computer}$
$TM = \text{Algorithm}$

FAST (P)
Polynomial

FAST (NP)
Unbounded Parallelism

SLOW
Exponential

Boolean Circuits

P = NP?

Chapters 28 & 29

Computability & Complexity

Algorithms & DS
- Approximation
- Randomized
- Distributed

Cryptography

Data
- ML/Al/DM/NLP
- Vision
- Graphics
- Comp. Finance

Networks
- Computers
- Social
- Data (e.g. www)

Robotics

Security

Programming Languages
- Compilers
- Distributed

Program Analysis
- Testing
- Verification

DB Systems

Parallel computing

Operating systems

Architecture

Graph theory

Linear Algebra

Probability Theory

Multivariate Calc.

Proof, logic

INDUCTION

Recursion

Struct. Induction

Sums, Asymptotics

Number theory

Graphs

Counting

Probability

Efficiency

Chapters

Introduction to Algorithms

 prerequisites

FOCS

Theory of Computing

Discrete Math

Creator: Malik Magdon-Ismail

Unsolvable Problems: 12 / 13

Epic Disasters →
...the high technology so celebrated today is essentially a mathematical technology.

“To err is human, but to really foul things up you need a computer.” – Paul Ehrlich
...the high technology so celebrated today is essentially a mathematical technology.

“To err is human, but to really foul things up you need a computer.” – Paul Ehrlich

- **Mariner rocket explodes (1962).** Formula into code bug resulted in no smoothing of deviations.
  - Luckily Stanislav “funny feeling in my gut” Petrov thought: “surely they’d use more missiles?”
- **Therac 25 (1985).** Concurrent programming bug killed patients through massive 100× radiation overdose.
- **AT&T Lines Go Dead (1990).** 75 million calls dropped (one line of buggy code in software upgrade).
- **Patriot missile defense fails (1991).** 28 soldiers dead, 100 injured (rounding error in scud-detection).
- **Pentium floating point long-division bug (1993).** Cost: $475 million – flawed division table.
- **Ariane rocket explosion (1996).** Cost: $500 million – overflow in 64-bit to 16-bit conversion.
- **Y2K (1999).** Cost: $500 billion spent because year was stored as 2 digits to save space.
- **Mars Climate Orbiter Crash (1998).** Cost: $125 million lost due to metric to imperial units bug.
- **Tesla Self-Driving Car (2016).** 1 dead. Auto-pilot didn’t “see” tractor-trailer.
- **Financial Disasters:** London Stock Exchange down due to single server bug (2009; billions of pounds of trading); Knight Capital computer glitch triggers stock sale (2012; 500 million lost and Knight’s value drops by 75%).
- **Airline Disasters:**
  - AirFrance 447 2009, 228 dead: pitot-tube failure feeds inconsistent data to programs which then panic pilot.
  - AdamAir 574, 2007, 102 dead: navigation system errors (and pilot errors).
  - Scottish RAF Chinook, 1994, 29 dead: faulty test program
  - AirFrance 296, 1988, 3 dead: altimeter bug.
  - IranAir 655, 1988, 290 dead: shot down by US Aegis combat system (misidentified as attacking military plane).
  - KoreanAir 007, 1983, 269 dead: autopilot took plane into Soviet airspace where it got shot down.
  - Boeing 737 Max, 2018, 2019, 346 dead: attack sensor + algorithm errors.
...the high technology so celebrated today is essentially a mathematical technology.

“To err is human, but to really foul things up you need a computer.” – Paul Ehrlich

- Mariner rocket explodes (1962). Formula into code bug resulted in no smoothing of deviations.
  - Luckily Stanislav “funny feeling in my gut” Petrov thought: “surely they’d use more missiles?”
- Therac 25 (1985). Concurrent programming bug killed patients through massive 100× radiation overdose.
- AT&T Lines Go Dead (1990). 75 million calls dropped (one line of buggy code in software upgrade).
- Y2K (1999). Cost: $500 billion spent because year was stored as 2 digits to save space.
- Financial Disasters: London Stock Exchange down due to single server bug (2009; billions of pounds of trading); Knight Capital computer glitch triggers stock sale (2012; 500 million lost and Knight’s value drops by 75%).
- Airline Disasters:
  - AirFrance 447 2009, 228 dead: pitot-tube failure feeds inconsistent data to programs which then panic pilot.
  - AdamAir 574, 2007, 102 dead: navigation system errors (and pilot errors).
  - Scottish RAF Chinook, 1994, 29 dead: faulty test program
  - AirFrance 296, 1988, 3 dead: altimeter bug.
  - IranAir 655, 1988, 290 dead: shot down by US Aegis combat system (misidentified as attacking military plane).
  - KoreanAir 007, 1983, 269 dead: autopilot took plane into Soviet airspace where it got shot down.
  - Boeing 737 Max, 2018, 2019, 346 dead: attack sensor + algorithm errors.

Software errors cost the U.S. $60 billion annually in rework, lost productivity and actual damages.

Put effort to make sure your program works fully correctly all the time.