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 \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.})
Last Time: Turing Machines

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

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

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

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

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

**input:** \langle G \rangle, the encoding of a graph G.
Intuitive notion of algorithm $\equiv$ Turing Machine
Solvable problem $\equiv$ Turing-decidable

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

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

($\langle G \rangle$ 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 $G$. 
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 \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 \( G \).
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 $\equiv$ Turing Machine

Solvable problem $\equiv$ Turing-decidable

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

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

($\langle G \rangle$ 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 $\equiv$ 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.)

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

**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 $G$.
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}

\begin{align*}
\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
\end{align*}

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

\begin{verbatim}
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.
\end{verbatim}
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.
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. $L_{\text{TM}}$: The language recognized by a Universal Turing Machine.
   - $L_{\text{TM}}$ 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.
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 $M$. 
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 $M$. 

---
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 $M$. 
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 $M$. 
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 $U_{TM}$.

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

$U_{TM}$ 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 $U_{TM}$.

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

$U_{TM}$ 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.
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.
Post’s Correspondence Problem (PCP) and HALFSUM

**PCP:** Consider 3 dominos: 

\[
\begin{array}{c|c|c}
\ hline
d_1 & d_2 & d_3 \\
\hline
0 & 01 & 110 \\
100 & 00 & 11 \\
\end{array}
\]
Post’s Correspondence Problem (PCP) and **HALFSUM**

**PCP:** Consider 3 dominos:  

<table>
<thead>
<tr>
<th></th>
<th>d₁</th>
<th>d₂</th>
<th>d₃</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>01</td>
<td>110</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
<td>11</td>
</tr>
</tbody>
</table>

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

← Top and bottom strings match.
That’s the goal.
Post’s Correspondence Problem (PCP) and HALFSUM

PCP: Consider 3 dominos: 

\[
\begin{array}{c}
d_1 \\
0 \\
100
\end{array} \quad \begin{array}{c}
d_2 \\
01 \\
00
\end{array} \quad \begin{array}{c}
d_3 \\
110 \\
11
\end{array}
\]

\[
d_3d_2d_3d_1 = \begin{array}{c}
110 \\
11 \\
10
\end{array} \quad \begin{array}{c}
011100 \\
001100 \\
100
\end{array} = \begin{array}{c}
110011100 \\
110011100
\end{array}
\]

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

INPUT: Dominos \(\{d_1, d_2, \ldots, d_n\}\). For example \([101, 011, 101]\).

TASK: Can one line up finitely many dominos so that the top and bottom strings match?
Post’s Correspondence Problem (PCP) and **HALFSUM**

**PCP:** Consider 3 dominos:

- \( d_1 = \begin{array}{c} 0 \\ 110 \\ 00 \end{array} \)
- \( d_2 = \begin{array}{c} 01 \\ 01 \end{array} \)
- \( d_3 = \begin{array}{c} 110 \end{array} \)

\[ d_3d_2d_3d_1 = \begin{array}{c} 110 \\ 01 \\ 110 \\ 0 \end{array} = \begin{array}{c} 110011100 \end{array} \]

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

**INPUT:** Dominos \( \{d_1, d_2, \ldots, d_n\} \). For example \( \{101, 011, 101\} \).

**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:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>=</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>=</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

**INPUT:** Dominos \{d_1, d_2, \ldots, d_n\}. For example \{\begin{array}{ccc} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \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 \(A = \{1, 3, 4, 9\}\).

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

**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 \(\frac{1}{2} \times \sum(S) = \frac{1}{2} \times (x_1 + x_2 + \cdots + x_n)\)?
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. ←program verification
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. ← program verification

What does Auto-Grade say for this program:

```
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**.  

$\text{Halts} = \begin{cases} 
\text{YES} & \text{if program halts} \\
\text{NO} & \text{if program infinitely loops}
\end{cases}$
Ultimate-Debugger

Wouldn’t it be nice to have the Ultimate-Debugger. ← solves the Halting Problem

\[
\text{Halts} \begin{cases} \begin{array}{l} n = 4; \\
\text{while}(n > 0)\{ \\
\quad \text{if}(n \text{ is not a sum of two primes})\{ \\
\quad \quad \text{print("Hello World!") and exit;} \\
\quad \} \\
\quad n \leftarrow n + 2; \\
\} 
\end{array} \end{cases}
\end{align*}
\]

\[
\text{Halts} = \begin{cases} \text{YES} & \text{if program halts} \\
\text{NO} & \text{if program infinitely loops} 
\end{cases}
\]

- 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?

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

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

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

\( U_{\text{TM}} \) 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 \( w \).
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 \( A_{\text{TM}}(\langle M \rangle \# \langle M \rangle) \)?
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.
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 \( A_{\text{TM}}(\langle M \rangle \# \langle M \rangle) \)?

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?

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

\( U_{TM} \) is a recognizer for \( L_{TM} \).

Is there a Turing Machine \( A_{TM} \) which decides \( L_{TM} \)?

- 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 \( A_{TM} \):

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

**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.** $A_{TM}$ does not exist ($L_{TM}$ Cannot be Solved)

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

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

$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D \rangle, \ldots$
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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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. \( A_{TM} \) 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 \( A_{TM}(\langle M_i \rangle \# \langle M_j \rangle) \).

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

<table>
<thead>
<tr>
<th>( A_{\text{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>( \cdots )</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>( \cdots )</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>( \cdots )</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 \( A_{\text{TM}}(\langle M_i \rangle \# \langle M_i \rangle) \).
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><strong>ACCEPT</strong></td>
<td><strong>ACCEPT</strong></td>
<td><strong>REJECT</strong></td>
<td><strong>ACCEPT</strong></td>
<td><strong>ACCEPT</strong></td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_2 \rangle$</td>
<td><strong>REJECT</strong></td>
<td><strong>REJECT</strong></td>
<td><strong>REJECT</strong></td>
<td><strong>ACCEPT</strong></td>
<td><strong>ACCEPT</strong></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><strong>REJECT</strong></td>
<td><strong>ACCEPT</strong></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,

$$⟨M_1⟩, ⟨M_2⟩, ⟨M_3⟩, ⟨M_4⟩, ⟨D⟩, \ldots$$

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

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

$D(⟨M_i⟩)$ does the opposite of $A_{TM}(⟨M_i⟩#⟨M_i⟩)$.
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 $A_{TM}(\langle M_i \rangle \# \langle M_j \rangle)$.

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

$D(\langle M_i \rangle)$ 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></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>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></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.** $A_{\text{TM}}$ 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 $A_{\text{TM}}(\langle M_i \rangle \# \langle M_j \rangle)$.

<table>
<thead>
<tr>
<th>$A_{\text{TM}}(\langle M_i \rangle # \langle M_j \rangle)$</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>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><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>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><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>$\vdots$</td>
</tr>
</tbody>
</table>

$D(\langle M_i \rangle)$ does the *opposite* of $A_{\text{TM}}(\langle M_i \rangle \# \langle M_i \rangle)$. 

---

Creator: Malik Magdon-Ismail

Unsolvable Problems: 9 / 13

ULTIMATE-DEBUGGER and AUTO-GRADE →
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>REJECT? $\ldots$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</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>ACCEPT</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>ACCEPT</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>ACCEPT</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>ACCEPT</td>
</tr>
<tr>
<td>$\langle D \rangle$</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>ACCEPT?</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</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. 😞
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 $L_{TM}$. 
No *general* program/algorithm to analyze *any* other program $M$ and tell if $M$ will accept or not a particular input.

Ultimate-Debugger and Auto-Grade Don’t Exist

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 $L_{TM}$.

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

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

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

**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.

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

**Exercise.** Show that **HalfSum** is solvable by giving a decider.
**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.

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:** $\langle M \rangle \# w$ 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.

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

**Exercise.** Show that **HALFSUM** is solvable by giving a decider.
The Landscape

DFA
(no external memory)
(regular expressions)
\{\ast 01\ast\}, \{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\},
\{ww^R\}

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

HalfSum
The Landscape

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

- **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

Unsolvable Problems: 11 / 13