Foundations of Computer Science
Lecture 27

Unsolvable Problems
A Powerful but Dangerous Technique
Analyzing Recursions and Recursions with Induction
Recursive Sets
Recursive Structures
Last Time: Turing Machines
Intuitive notion of algorithm \equiv \text{Turing Machine}

Solvable problem \equiv \text{Turing-decidable}
Last Time: Turing Machines

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

$L = \{\langle G \rangle \mid G \text{ is connected}\}$
Last Time: Turing Machines

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

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

\[ \langle G \rangle \]

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

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

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

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

($\langle G \rangle$ 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

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

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

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

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.)
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.)
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.)
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.)
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 node in $G$.
2: **REPEAT:** Find an edge in $G$ between a marked and an unmarked node.
   Mark the unmarked node or **GOTO** step 3 if there is no such edge.
3: **REJECT** if there is an unmarked node remaining in $G$; otherwise **ACCEPT**.
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 node in $G$.
2. **REPEAT:** Find an edge in $G$ between a marked and an unmarked node.
   - Mark the unmarked node or **GOTO** step 3 if there is no such edge.
3. **REJECT** if there is an unmarked node 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.
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.)

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

**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 node in \( G \).
2. **REPEAT:** Find an edge in \( G \) between a marked and an unmarked node.
   Mark the unmarked node or **GOTO** step 3 if there is no such edge.
3. **REJECT** if there is an unmarked node 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?
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 \( G \).

1. Check that \( \langle G \rangle \) is a valid encoding of a graph and mark the first node in \( G \).
2. **REPEAT:** Find an edge in \( G \) between a marked and an unmarked node.
   - Mark the unmarked node or **GOTO** step 3 if there is no such edge.
3. **REJECT** if there is an unmarked node 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?
Programmable Turing Machine: Universal Turing Machine
A Turing Machine $M$ has a binary encoding $\langle M \rangle$. 
Programmable Turing Machine: Universal Turing Machine

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 $U_{TM}$. 
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 $U_{\text{TM}}$.

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

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

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

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

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

\( U_{TM} \) 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 $U_{\text{TM}}$.

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

$U_{\text{TM}}$ 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 $M$. 

Unsolvable Problems: 4 / 13
PCP and HALFSUM →
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.
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.
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.
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.
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 HALF\textsc{Sum}
Post’s Correspondence Problem (PCP) and \textbf{HALFSUM}

\textbf{PCP}: Consider 3 dominos: 

\begin{tabular}{c c c}
\hline
$d_1$ & $d_2$ & $d_3$ \\
\hline
0 & 01 & 110 \\
100 & 00 & 11 \\
\hline
\end{tabular}
Post’s Correspondence Problem (PCP) and \textbf{HALF\text{SUM}}

\textbf{PCP:} Consider 3 dominos:

\begin{tabular}{ccc}
$d_1$ & $d_2$ & $d_3$ \\
0 & 01 & 110 \\
100 & 00 & 11
\end{tabular}

\[d_3 = \begin{array}{c}
110 \\
11
\end{array}\]
Post’s Correspondence Problem (PCP) and **HALF$$\text{SUM}$$**

**PCP:** Consider 3 dominos: 

\[
\begin{array}{c}
\begin{array}{c}
\hline
0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 \\
\hline
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\hline
0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 \\
\hline
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\hline
1 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 \\
\hline
\end{array}
\end{array}
\]

\[
d_3d_2 = \begin{array}{c}
\begin{array}{c}
\hline
1 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 \\
\hline
\end{array}
\end{array}
\]

\[
= \begin{array}{c}
\begin{array}{c}
\hline
1 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 \\
\hline
\end{array}
\end{array}
\]

\[
= \begin{array}{c}
\begin{array}{c}
\hline
1 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 \\
\hline
\end{array}
\end{array}
\]
Post’s Correspondence Problem (PCP) and \textsc{HalfSum}

**PCP:** Consider 3 dominos:

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

\[
d_3d_2d_3 = \begin{array}{c|c|c}
110 & 01 & 110 \\
11 & 00 & 11 \\
\end{array}
\]
Post's Correspondence Problem (PCP) and **HALF**SUM

**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>110</td>
<td>01</td>
<td>110</td>
</tr>
<tr>
<td>100</td>
<td>00</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

$$d_3d_2d_3d_1 = \begin{array}{cccc}
110 & 01 & 110 & 0 \\
11 & 00 & 11 & 100 \\
\end{array}$$
Post’s Correspondence Problem (PCP) and **HALFSUM**

**PCP:** Consider 3 dominos:

\[
\begin{array}{ccc}
\hline
& d_1 & d_2 & d_3 \\
\hline
\text{0} & \text{01} & \text{110} \\
\text{100} & \text{00} & \text{11} \\
\hline
\end{array}
\]

\[d_3 d_2 d_1 = \begin{array}{cccc}
\text{110} & \text{01} & \text{110} & \text{0} \\
\text{11} & \text{00} & \text{11} & \text{100} \\
\end{array} = \begin{array}{c}
\text{110011100} \\
\text{110011100} \\
\end{array}\]
Post’s Correspondence Problem (PCP) and HALFSUM

**PCP:** Consider 3 dominos:

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>100</td>
<td>00</td>
<td>11</td>
</tr>
</tbody>
</table>

$$d_3d_2d_3d_1 = \begin{array}{cccc}
11 & 01 & 110 & 0 \\
11 & 00 & 11 & 100 \\
\end{array} = \begin{array}{c}
110011100110011100 \\
110011100110011100 \\
\end{array}$$

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

**PCP:** Consider 3 dominos:

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

\[
d_3 d_2 d_3 d_1 = 110 01 110 011001100110011100 = 11001100110011100 \\
11 00 11 100 \\
\hline
\]

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

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

Creator: Malik Magdon-Ismail
Post’s Correspondence Problem (PCP) and HALFSUM

**PCP:** Consider 3 dominos: \( d_1 \), \( d_2 \), \( d_3 \):

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

\[
d_3d_2d_3d_1 = \begin{array}{c}
110 & 01 & 110 & 0 \\
11 & 00 & 11 & 100
\end{array} = \begin{array}{c}
110011100110011100 \\
110011100110011100
\end{array}
\]

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

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

**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:

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

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

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

**INPUT:** Dominos \( \{d_1, d_2, \ldots, d_n\} \). For example \( \begin{array}{c}
10 \\
101 \\
01 \\
11 \\
101 \\
011
\end{array} \).

**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 \( A = \{1, 3, 4, 9\} \).
Post’s Correspondence Problem (PCP) and **HALF**SUM

**PCP:** Consider 3 dominos:

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

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

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

**INPUT:** Dominos \(\{d_1, d_2, \ldots, d_n\}\). For example, \(\begin{bmatrix} 10, 011, 011 \end{bmatrix}\).

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

**HALF**SUM: 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, \(S = \{1, 1, 1, 3, 4, 4, 5, 6, 9\}\).
Post’s Correspondence Problem (PCP) and \textbf{HALFSum}

\textbf{PCP:} Consider 3 dominos:
\[
\begin{array}{ccc}
\hline
& d_1 & d_2 & d_3 \\
\hline
0 & 01 & 110 \\
100 & 00 & 11 \\
\hline
\end{array}
\]
\[
d_3d_2d_3d_1 = 110011100110011100
\]
← Top and bottom strings match. That’s the goal.

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

\textbf{TASK:} Can one line up finitely many dominos so that the top and bottom strings match?

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

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

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

\textbf{TASK:} Is there a subset whose sum is \( \frac{1}{2} \times \text{sum}(S) = \frac{1}{2} \times (x_1 + x_2 + \cdots + x_n) ? \)
Your first CS assignment: Write a program to print “Hello World!” and halt.
Your first CS assignment: Write a program to print “Hello World!” and halt.

CS1: 700+ submissions!
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**

**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 it’s correct. ←program verification
**Auto-Grade**

**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:

```plaintext
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**. \[\rightarrow\text{solves the } Halting\text{ Problem}\]
Wouldn’t it be nice to have the **ULTIMATE-DEBUGGER**. ↓ solves the *Halting Problem*

\[
\text{Halts} = \begin{cases} 
 n = 4; \\
 \text{while}(n > 0)\{ \\
 \quad \text{if}(n \text{ is not a sum of two primes})\{ \\
 \quad \quad \text{print}(\text{"Hello World!"}) \text{ and exit}; \\
 \quad \} \\
 \quad n \leftarrow n + 2; \\
 \} 
\end{cases}
\]
Wouldn’t it be nice to have the **Ultimate-Debugger**.  

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

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

\[
\text{Halts} = \begin{cases} 
\text{YES} & \text{if program halts} \\
\text{NO} & \text{if program infinitely loops}
\end{cases}
\]
Wouldn’t it be nice to have the **Ultimate-Debugger**. ← solves the *Halting Problem*

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

- We can grade the students program correctly.
Ultimate-Debugger

 Wouldn’t it be nice to have the **Ultimate-Debugger**. \[ \text{解决 Halting Problem} \]

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

- We can grade the students program correctly.
- We can solve Goldbach’s conjecture.
Wouldn’t it be nice to have the **Ultimate-Debugger**.  

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

- 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.
The Language of Successfully Terminating Programs
The Language of Successfully Terminating Programs

\[ L_{TM} = \{ \langle M \rangle \# w \mid M \text{ is a Turing Machine and } M \text{ accepts } w \}. \]
The Language of Successfully Terminating Programs

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

\( U_{\text{TM}} \) is a recognizer for \( \mathcal{L}_{\text{TM}} \).
The Language of Successfully Terminating Programs

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

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

Is there a Turing Machine \( A_{\text{TM}} \) which decides \( \mathcal{L}_{\text{TM}} \)?
The Language of Successfully Terminating Programs

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

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

\( U_{TM} \) 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 \( w \).
The Language of Successfully Terminating Programs

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

\(U_{\text{TM}}\) 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 \(w\).
- What do these mean: \(M(\langle M \rangle)\) and \(A_{\text{TM}}(\langle M \rangle \# \langle M \rangle)\)?
The Language of Successfully Terminating Programs

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

\( U_{\text{TM}} \) 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 \( w \).
- What do these mean: \( M(\langle M \rangle) \) and \( A_{\text{TM}}(\langle M \rangle \# \langle M \rangle) \)?

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

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

The Language of Successfully Terminating Programs

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

\( U_{\text{TM}} \) 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 \( w \).
- What do these mean: \( M(\langle M \rangle) \) and \( A_{\text{TM}}(\langle M \rangle \# \langle M \rangle) \)?

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

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

**input:** \( \langle M \rangle \) where \( M \) is a Turing Machine.
The Language of Successfully Terminating Programs

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

\( U_{\text{TM}} \) 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 \( w \).
- What do these mean: \( M(\langle M \rangle) \) and \( A_{\text{TM}}(\langle M \rangle \# \langle M \rangle) \)?

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

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

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

1. Run \( A_{\text{TM}} \) with input \( \langle M \rangle \# \langle M \rangle \).
The Language of Successfully Terminating Programs

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

\( U_{\text{TM}} \) 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 \( w \).
- What do these mean: \( M(\langle M \rangle) \) and \( A_{\text{TM}}(\langle M \rangle \# \langle M \rangle) \)?

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

\[
D_{\text{TM}} = \text{“Diagonal” Turing Machine derived from } A_{\text{TM}} \text{ (the decider for } \mathcal{L}_{\text{TM}}) \\
\text{input: } \langle M \rangle \text{ where } M \text{ is a Turing Machine.} \\
\begin{align*}
1: & \text{ Run } A_{\text{TM}} \text{ with input } \langle M \rangle \# \langle M \rangle. \\
2: & \text{ If } A_{\text{TM}} \text{ accepts then REJECT; otherwise (} A_{\text{TM}} \text{ rejects) ACCEPT}
\end{align*}
\]

\( D_{\text{TM}} \) does the opposite of \( A_{\text{TM}} \). Is \( D_{\text{TM}} \) a decider?
Theorem. \( A_{TM} \) does not exist (\( L_{TM} \) Cannot be Solved)
Theorem. $A_{TM}$ does not exist ($L_{TM}$ Cannot be Solved)

$A_{TM}$ exists $\rightarrow D_{TM}$ exists.
Theorem. $A_{TM}$ does not exist ($\mathcal{L}_{TM}$ Cannot be Solved)

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

$D_{TM}$ exists means it will appear on the list of all Turing Machines,

$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D_{TM} \rangle, \ldots$
Theorem. $A_{TM}$ does not exist ($L_{TM}$ Cannot be Solved)

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

$D_{TM}$ exists means it will appear on the list of all Turing Machines,

$$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D_{TM} \rangle, \ldots$$

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

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

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

\( D_{TM} \) exists means it will appear on the list of all Turing Machines,

\[ \langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D_{TM} \rangle, \ldots \]

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_{TM} \rangle )</th>
<th>( \vdots )</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_{TM} \rangle )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( \langle M_2 \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_{TM} \rangle )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( \langle M_3 \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_{TM} \rangle )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( \langle M_4 \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_{TM} \rangle )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( \langle D_{TM} \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_{TM} \rangle )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( \vdots )</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_{TM} \rangle )</td>
<td>( \vdots )</td>
</tr>
</tbody>
</table>
Theorem. \( A_{TM} \) does not exist (\( L_{TM} \) Cannot be Solved)

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

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

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_{TM} \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></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_{TM} \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 $\rightarrow D_{TM}$ exists.

$D_{TM}$ exists means it will appear on the list of all Turing Machines,

\[
\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D_{TM} \rangle, \ldots
\]

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>$\langle D_{TM} \rangle$</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>$\langle D_{TM} \rangle$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REJECT</td>
<td></td>
</tr>
<tr>
<td>$\vdots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$D_{TM}(\langle M_i \rangle)$ 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 $\rightarrow D_{TM}$ exists.

$D_{TM}$ exists means it will appear on the list of all Turing Machines,

$$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D_{TM} \rangle, \ldots$$

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>$\langle D_{TM} \rangle$</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>$\langle D_{TM} \rangle$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>REJECT</td>
</tr>
<tr>
<td>$\vdots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$D_{TM}(\langle M_i \rangle)$ 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 $\rightarrow D_{TM}$ exists.

$D_{TM}$ exists means it will appear on the list of all Turing Machines,

$$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D_{TM} \rangle, \ldots$$

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>$\langle D_{TM} \rangle$</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>$\langle D_{TM} \rangle$</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>

$D_{TM}(\langle M_i \rangle)$ 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 $\rightarrow D_{TM}$ exists.

$D_{TM}$ exists means it will appear on the list of all Turing Machines,

\[ \langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D_{TM} \rangle, \ldots \]

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>$\langle D_{TM} \rangle$</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>$\langle D_{TM} \rangle$</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>

$D_{TM}(\langle M_i \rangle)$ 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 $\rightarrow D_{TM}$ exists.

$D_{TM}$ exists means it will appear on the list of all Turing Machines,

$$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D_{TM} \rangle, \ldots$$

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>$\langle D_{TM} \rangle$</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>$\ldots$</td>
</tr>
<tr>
<td>$\langle D_{TM} \rangle$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>

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

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 $\rightarrow D_{TM}$ exists.

$D_{TM}$ exists means it will appear on the list of all Turing Machines,

$$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D_{TM} \rangle, \ldots$$

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>$\langle D_{TM} \rangle$</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>REJECT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle D_{TM} \rangle$</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>

$D_{TM}(\langle M_i \rangle)$ 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 $\rightarrow D_{TM}$ exists.

$D_{TM}$ exists means it will appear on the list of all Turing Machines,

$$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D_{TM} \rangle, \ldots$$

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>$\langle D_{TM} \rangle$</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>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>$\langle D_{TM} \rangle$</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>

$D_{TM}(\langle M_i \rangle)$ 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 \( \rightarrow \) \( D_{\text{TM}} \) exists.

\( D_{\text{TM}} \) exists means it will appear on the list of all Turing Machines,

\[ \langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D_{\text{TM}} \rangle, \ldots \]

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

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

\( D_{\text{TM}}(\langle M_i \rangle) \) 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)

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

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

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>$\langle D_{TM} \rangle$</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><strong>ACCEPT</strong></td>
<td><strong>ACCEPT</strong></td>
<td><strong>REJECT</strong></td>
<td><strong>REJECT</strong></td>
<td><strong>ACCEPT</strong></td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_4 \rangle$</td>
<td><strong>ACCEPT</strong></td>
<td><strong>REJECT</strong></td>
<td><strong>REJECT</strong></td>
<td><strong>REJECT</strong></td>
<td><strong>ACCEPT</strong></td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle D_{TM} \rangle$</td>
<td><strong>REJECT</strong></td>
<td><strong>ACCEPT</strong></td>
<td><strong>ACCEPT</strong></td>
<td><strong>ACCEPT</strong></td>
<td><strong>REJECT?</strong></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>

$D_{TM}(\langle M_i \rangle)$ 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 $\rightarrow D_{TM}$ exists.

$D_{TM}$ exists means it will appear on the list of all Turing Machines,

$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D_{TM} \rangle, \ldots$

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

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

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

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 general program/algorithm to analyze any other program $M$ and tell if $M$ will accept or not a particular input. 😞

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

\[ A_{TM} = \text{Turing Machine derived from } H_{TM} \text{ (the decider for } L_{\text{HALT}}) \]
No general program/algorithm to analyze any other program $M$ and tell if $M$ will accept or not a particular input.

Suppose **Ultimate-Debugger** $H_{\text{TM}}$ 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 $M$. 
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.
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.
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 $M$.

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.
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.
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 \( L_{TM} \).

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

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)
\{ *01*, \{0\^{3n+1}\} \}
The Landscape

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

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

DFA
(no external memory)
(regular expressions)
\{\star 01\star\}, \{0^n1^n\}, \{w R\}, \{0^n1^n0^n\}

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

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

HalfSum
The Landscape

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

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

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

TM-Recognizer 
\mathcal{L}_TM
Ultimate-Debugger
Auto-Grade
PCP

Unsolvable Problems: 11 / 13
The Landscape

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

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

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

TM-Recognizer
\mathcal{L}_{TM}
Ultimate-Debugger
Auto-Grade
PCP

HalfSum

Non-Recognizable
\overline{\mathcal{L}_{TM}}, \overline{\mathcal{L}_{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
The Path Forward: Focus on Decidable Problems
The Path Forward: Focus on Decidable Problems
The Path Forward: Focus on Decidable Problems
The Path Forward: Focus on Decidable Problems

FOCS

Theory of Computing
- CFG Parsing
- DFA RegExp

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
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

Creator: Malik Magdon-Ismail

Unsolvable Problems: 12 / 13

Epic Disasters →
The Path Forward: Focus on Decidable Problems

Decider
\( U_{TM} = \text{computer} \)
\( TM = \text{Algorithm} \)

FOCS

Theory of Computing

Decider
\( U_{TM} = \text{computer} \)
\( TM = \text{Algorithm} \)

DFA
RegExp

CFG
Parsing

Proof, logic
INDUCTION

DISCRETE MATH

Recursion
Struct. Induction

Sums, Asymptotics

Number theory

Graphs

Counting

Probability

Graph theory
Linear Algebra
Probability Theory
Multivariate Calc.
The Path Forward: Focus on Decidable Problems

Decider

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

$TM = \text{Algorithm}$

效率

FAST (P)

Polynomial

FOCS

理论与计算

CFG Parsing

DFA RegExp

理论

多变量积分

线性代数

概率论

图论

概率

FOCS

离散数学

归纳

归纳

递归

结构归纳

和，非零函数

数论

图

计数

概率

多变量积分

图论

线性代数

概率论
The Path Forward: Focus on Decidable Problems

FOCS

Theory of Computing

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

SLOW

Exponential

Graph theory

Linear Algebra

Probability Theory

Multivariate Calc.
The Path Forward: Focus on Decidable Problems

Theory of Computing

- Decider
  - $U_{tm} =$ computer
  - TM = Algorithm

- CFG
  - Parsing

- DFA
  - RegExp

Proof, logic

- INDUCTION
  - Recursion
  - Struct. Induction

Discrete Math

- Sums, Asymptotics
- Number theory
- Graphs
- Counting
- Probability

FOCS

Efficiency

- FAST (P)
  - Polynomial

- FAST (NP)
  - Unbounded Parallelism

- SLOW
  - Exponential

Unsolvable Problems: 12 / 13

Creator: Malik Magdon-Ismail

Epic Disasters →
The Path Forward: Focus on Decidable Problems

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

FOCS

Theory of Computing

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

Induction

Recursion
Struct. Induction

Sums, Asymptotics

Number theory

Graphs

Counting

Probability

Efficiency

FAST (P)
Polynomial

FAST (NP)
Unbounded Parallelism

SLOW
Exponential

P = NP?
The Path Forward: Focus on Decidable Problems

Decider

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

$TM = \text{Algorithm}$

FOCS

Theory of Computing

Decider

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

$TM = \text{Algorithm}$

Fast (P)
Polynomial

Fast (NP)
Unbounded Parallelism

SLOW
Exponential

Boolean Circuits

Proof, logic

INDUCTION

Recursion
Struct. Induction

Sums, Asymptotics

Number theory

Graphs

Counting

Probability

Graph theory

Linear Algebra

Probability Theory

Multivariate Calc.

Unsolvable Problems: 12 / 13

Epic Disasters
The Path Forward: Focus on Decidable Problems

Decider
\[ U_{TM} = \text{computer} \]
\[ TM = \text{Algorithm} \]

Theory of Computing

Decision

CFG Parsing

DFA RegExp

Proof, logic

Induction

Recursion

Struct. Induction

Sums, Asymptotics

Number theory

Graphs

Counting

Probability

FOCS

Computability & Complexity

FAST (P)

Polynomial

FAST (NP)

Unbounded Parallelism

SLOW

Exponential

Boolean Circuits

Chapters

28 & 29

P = NP?

Graph theory

Linear Algebra

Probability Theory

Multivariate Calc.
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

Proof, logic
INDUCTION

Recursion
Struct. Induction

Sums, Asymptotics

Number theory

Graphs

Counting

Probability

Graph theory
Linear Algebra
Probability Theory
Multivariate Calc.

Multivariate Calc.

FAST (P)
Polynomial

FAST (NP)
Unbounded Parallelism

SLOW
Exponential

Boolean Circuits

P = NP?

Chapters
28 & 29

Computability & Complexity

Introduction to Algorithms

Principles of Software

Computer Organization

Creator: Malik Magdon-Ismail

Unsolvable Problems: 12 / 13

Epic Disasters
The Path Forward: Focus on Decidable Problems

FOCS

Theory of Computing

Computability & Complexity

P = NP?

Decider

UTm = computer

TM = Algorithm

FAST (P)

Polynomial

FAST (NP)

Unbounded Parallelism

SLOW

Exponential

Boolean Circuits

Chapters 28 & 29

Cryptography

Algorithms & DS

- Approximation
- Randomized
- Distributed

Data

- ML/AI/DM/NLP
- Vision
- Graphics
- Comp. Finance

Networks

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

Robotics

Security

Programming Languages

- Compilers
- Distributed

Program Analysis

- Testing
- Verification

Theory

Algorithms

AI

Introduction to Algorithms

Principles of Software

Computer Organization

FOCS

Discrete Math

Proof, logic

INDUCTION

Graph theory

Linear Algebra

Probability Theory

Multivariate Calc.

Recursion

Struct. Induction

Sums, Asymptotics

Number theory

Graphs

Counting

Probability

Unsolvable Problems: 12 / 13

Epic Disasters

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

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

CFG
Parsing

DFA
RegExp

Graph theory
Linear Algebra
Probability Theory
Multivariate Calc.

Proof, logic
INDUCTION

Recursion
Struct. Induction

Sums, Asymptotics

Number theory

Graphs

Counting

Probability

Decider
FM = computer
TM = Algorithm

FAST (P)
Polynomial

FAST (NP)
Unbounded Parallelism

SLOW
Exponential

Boolean Circuits

Chapters 28 & 29

P = NP?

Computability & Complexity

Algorithms & DS
- Approximation
- Randomized
- Distributed

Cryptography

Data
- ML/AI/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

Unsolvable Problems: 12 / 13

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

Graph theory
Linear Algebra
Probability Theory
Multivariate Calc.

Induction
Recursion
Struct. Induction
Sums, Asymptotics
Number theory
Graphs
Counting
Probability

Theory of Computing

Discrete Math

Computability & Complexity

\( P = \text{NP} \)?

Chapters 28 & 29

Algorithms & DS
- Approximation
- Randomized
- Distributed

Cryptography

Data
- ML/AI/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

Theory
Algorithms
AI

Introduction to Algorithms

Principles of Software

Software Systems

Creator: Malik Magdon-Ismail

Unsolvable Problems: 12 / 13

Epic Disasters →
The Path Forward: Focus on Decidable Problems

FOCS

Theory of Computing
- Decider
- $U_{tm} =$ computer
- TM = Algorithm
- CFG
- Parsing
- DFA
- RegExp

Discrete Math
- Proof, logic
- INDUCTION
- Recursion
- Struct. Induction
- Sums, Asymptotics
- Number theory
- Graphs
- Counting
- Probability

FOCS

Computability & Complexity
- FAST (P) Polynomial
- FAST (NP) Unbounded Parallelism
- SLOW Exponential

P = NP?

Chapters 28 & 29

Introduction to Algorithms
- Algorithms & DS
  - Approximation
  - Randomized
  - Distributed
- Cryptography
- Data
  - ML/Al/DM/NLP
  - Vision
  - Graphics
  - Comp. Finance

Computers, Systems
- Networks
  - Computers
  - Social
  - Data (e.g. www)
- Robotics
- Security
- Programming Languages
  - Compilers
  - Distributed
- Program Analysis
  - Testing
  - Verification
- DB Systems
- Parallel computing
- Operating systems
- Architecture

Theory Algorithms AI

Software Systems

Robotics

Security

Programming Languages
- Compilers
- Distributed

Program Analysis
- Testing
- Verification

DB Systems
- Parallel computing
- Operating systems
- Architecture

Unsolvable Problems: 12 / 13
Epic Disasters →

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

FOCS

Theory of Computing
- Decider
  $U_{tm} = \text{computer}$
  $TM = \text{Algorithm}$
- CFG
  Parsing
- DFA
  RegExp
- Graph theory
- Linear Algebra
- Probability Theory
- Multivariate Calc.
- Proof, logic
  INDUCTION
- Recursion
  Struct. Induction
- Sums, Asymptotics
- Number theory
- Graphs
- Counting
- Probability
- FAST (P)
  Polynomial
- FAST (NP)
  Unbounded Parallelism
- SLOW
  Exponential
- Boolean Circuits

Computability & Complexity
- Algorithms & DS
  - Approximation
  - Randomized
  - Distributed
- Cryptography
- Data
  - ML/AI/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

Unsolvable Problems: 12 / 13
Epic Disasters →

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?

Computability & Complexity

Algorithms & DS
- Approximation
- Randomized
- Distributed

Cryptography

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

Chapters 28 & 29

Introduction

Theory to Algorithms

FOCS

Theory of Computing

Discrete Math

FOCS

Unsolvable Problems: 12 / 13

Epic Disasters

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

FOCS

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

Graph theory
Linear Algebra
Probability Theory
Multivariate Calc.

Theory of Computing

Computability & Complexity

Algorithms & DS
- Approximation
- Randomized
- Distributed

Cryptography

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

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

Robotics

Security

Programming Languages
- Compilers
- Distributed

Program Analysis
- Testing
- Verification

DB Systems

Parallel computing

Operating systems

Architecture

Chapters 28 & 29

P = NP?

Introduction to Algorithms

Principles of Software

Computer Organization

Software Systems

Theory Algorithms

AI

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 $100 \times$ radiation overdose.
- AT&T Lines Go Dead (1990). 75 million calls dropped (one line of buggy code in software upgrade).
- Y2K (1999). Cost: $500 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.
...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 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.
- **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).
  - KoreanAir 801, 1997, **228 dead**: ground proximity warning system bug.
  - AeroPerú 603, 1996, **70 dead**: altimeter failures.
  - 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.
- **Software errors cost the U.S. $60 billion annually in rework, lost productivity and actual damages.**
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 $100 \times$ radiation overdose.
- **AT&T Lines Go Dead** (1990). 75 million calls dropped (one line of buggy code in software upgrade).
- **Y2K** (1999). Cost: $500 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.
- **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, 2008, **154 dead**: malware virus.
  ▶ KoreanAir 801, 1997, **228 dead**: ground proximity warning system bug.
  ▶ AeroPerú 603, 1996, **70 dead**: altimeter failures.
  ▶ AirFrance 296, 1988, **3 dead**: altimeter bug.
  ▶ Iranian Air 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.
- **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.