Infinite Loops

What happens if the input is 01?

Turing Machine $M$ is a recognizer for language $\mathcal{L}(M)$:

$$w \in \mathcal{L}(M) \iff M(w) = \text{halt with a } \text{YES};$$

$$w \not\in \mathcal{L}(M) \iff M(w) = \text{halt with a } \text{NO} \text{ or loop forever.}$$

Turing Machine $M$ is a decider for language $\mathcal{L}(M)$:

$$w \in \mathcal{L}(M) \iff M(w) = \text{halt with a } \text{YES};$$

$$w \not\in \mathcal{L}(M) \iff M(w) = \text{halt with a } \text{NO}.$$
**Mathematical Description of a Turing Machine**

1. **States** $Q$. The first state is the start state, the halting states are $A, R$.
2. **Symbols** $\Sigma$. By default these are $\{*, 0, 1, \#, \}$.  
3. **Machine-level transition instructions.** Each instruction has the form
   
   \[
   \{\text{state}\}{\text{read-symbol}}{\text{next-state}}{\text{written-symbol}}{\text{move}}
   \]

   The instructions map each $(\text{state}, \text{symbol})$ pair to a $(\text{state}, \text{symbol}, \text{move})$ triple and thus form a *transition function* $\delta : Q \times \Sigma \mapsto Q \times \Sigma \times \{L, R, S\}$. 

Mathematical Description of a Turing Machine

1. **States** $Q$. The first state is the start state, the halting states are $A, R$.
2. **Symbols** $\Sigma$. By default these are $\{\ast, 0, 1, \sqcup, \#\}$.
3. **Machine-level transition instructions.** Each instruction has the form
   
   $\{\text{state}\}{\text{read-symbol}}{\text{next-state}}{\text{written-symbol}}{\text{move}}$

   The instructions map each (state,symbol) pair to a (state,symbol,move) triple and thus form a *transition function* $\delta : Q \times \Sigma \mapsto Q \times \Sigma \times \{L, R, S\}$.
Mathematical Description of a Turing Machine

1. **States** $Q$. The first state is the start state, the halting states are $A,R$.
2. **Symbols** $\Sigma$. By default these are $\{*, 0, 1, \_, #\}$.
3. **Machine-level transition instructions.** Each instruction has the form
   
   $$\{\text{state}\}\{\text{read-symbol}\}\{\text{next-state}\}\{\text{written-symbol}\}\{\text{move}\}$$

   The instructions map each (state,symbol) pair to a (state,symbol,move) triple and thus form a transition function $\delta : Q \times \Sigma \mapsto Q \times \Sigma \times \{L,R,S\}$.

**States.** $\{q_0, q_1, A, E\}$
Mathematical Description of a Turing Machine

1. **States** $Q$. The first state is the start state, the halting states are $A, R$.
2. **Symbols** $\Sigma$. By default these are $\{*, 0, 1, \sqcup, \#\}$.
3. **Machine-level transition instructions.** Each instruction has the form
   \[ \{\text{state}\}\{\text{read-symbol}\}\{\text{next-state}\}\{\text{written-symbol}\}\{\text{move}\} \]
   The instructions map each (state,symbol) pair to a (state,symbol,move) triple and thus form a *transition function* $\delta : Q \times \Sigma \mapsto Q \times \Sigma \times \{L, R, S\}$.

- **States.** $\{q_0, q_1, A, E\}$
- **Symbols.** $\{*, 0, 1, \sqcup, \#\}$
Encoding a Turing Machine as A Bit-String

Mathematical Description of a Turing Machine

1. **States** $Q$. The first state is the start state, the halting states are $A, R$.
2. **Symbols** $\Sigma$. By default these are $\{*, 0, 1, \_, #\}$.
3. **Machine-level transition instructions.** Each instruction has the form
   
   $\{\text{state}\}\{\text{read-symbol}\}\{\text{next-state}\}\{\text{written-symbol}\}\{\text{move}\}$

   The instructions map each (state,symbol) pair to a (state,symbol,move) triple and thus form a transition function $\delta : Q \times \Sigma \mapsto Q \times \Sigma \times \{L, R, S\}$.

**States.** $\{q_0, q_1, A, E\}$

**Symbols.** $\{*, 0, 1, \_, #\}$

**Machine-level transition instructions.**

$\{q_0\}\{\_\}\{q_0\}\{\_\}\{R\}$
$\{q_0\}\{1\}\{q_0\}\{1\}\{R\}$
$\{q_0\}\{0\}\{q_1\}\{0\}\{R\}$
$\{q_0\}\{\_\}\{E\}\{\_\}\{S\}$
$\{q_0\}\{\_\}\{E\}\{\_\}\{S\}$
$\{q_1\}\{1\}\{q_0\}\{1\}\{L\}$
$\{q_1\}\{0\}\{A\}\{0\}\{S\}$
$\{q_1\}\{\_\}\{A\}\{\_\}\{S\}$
$\{q_1\}\{\_\}\{A\}\{\_\}\{S\}$
Mathematical Description of a Turing Machine
1. **States** \( Q \). The first state is the start state, the halting states are \( A,R \).
2. **Symbols** \( \Sigma \). By default these are \(*, 0, 1, \_\, \#, \)\.
3. **Machine-level transition instructions.** Each instruction has the form

\[
\text{state}\{\text{read-symbol}\}\{\text{next-state}\}\{\text{written-symbol}\}\{\text{move}\}
\]

The instructions map each (state,symbol) pair to a (state,symbol,move) triple and thus form a *transition function* \( \delta : Q \times \Sigma \mapsto Q \times \Sigma \times \{L,R,S\} \).

### States. \( \{q_0, q_1, A, E\} \)

### Symbols. \( \{*, 0, 1, \_, \#\} \)

### Machine-level transition instructions.

\[
\begin{align*}
q_0\{\_\}\{q_0\} & \{\_\}\{R\} \\
q_0\{1\}\{q_0\} & \{1\}\{R\} \\
q_0\{0\}\{q_1\} & \{0\}\{R\} \\
q_0\{\#\}\{E\} & \{\#\}\{S\} \\
q_0\{\_\}\{E\} & \{\_\}\{S\} \\
q_1\{1\}\{q_0\} & \{1\}\{L\} \\
q_1\{0\}\{A\} & \{0\}\{S\} \\
q_1\{\#\}\{A\} & \{\#\}\{S\} \\
q_1\{\_\}\{A\} & \{\_\}\{S\}
\end{align*}
\]

The description of a Turing Machine is a *finite* binary string.
Mathematical Description of a Turing Machine

1. **States** $Q$. The first state is the start state, the halting states are $A, R$.
2. **Symbols** $\Sigma$. By default these are $\{*, 0, 1, \_, \#\}$.
3. **Machine-level transition instructions.** Each instruction has the form
   $$\{\text{state}\}\{\text{read-symbol}\}\{\text{next-state}\}\{\text{written-symbol}\}\{\text{move}\}$$
   The instructions map each (state,symbol) pair to a (state,symbol,move) triple and thus form a transition function $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R, S\}$.

**States.** $\{q_0, q_1, A, E\}$

**Symbols.** $\{*, 0, 1, \_, \#\}$

**Machine-level transition instructions.**
- $\{q_0\}\{\*\}\{q_0\}\{\*\}\{R\}$
- $\{q_0\}\{1\}\{q_0\}\{1\}\{R\}$
- $\{q_0\}\{0\}\{q_1\}\{0\}\{R\}$
- $\{q_0\}\{\#\}\{E\}\{\#\}\{S\}$
- $\{q_0\}\{\_\}\{E\}\{\_\}\{S\}$
- $\{q_1\}\{1\}\{q_0\}\{1\}\{L\}$
- $\{q_1\}\{0\}\{A\}\{0\}\{S\}$
- $\{q_1\}\{\#\}\{A\}\{\#\}\{S\}$
- $\{q_1\}\{\_\}\{A\}\{\_\}\{S\}$

The description of a Turing Machine is a *finite* binary string.

Turing machines are countable and can be listed: $\{M_1, M_2, \ldots\}$. 
Mathematical Description of a Turing Machine

1. **States** $Q$. The first state is the start state, the halting states are A,R.
2. **Symbols** $Σ$. By default these are $\{$*, 0, 1, UnderTest, #$.\}$
3. **Machine-level transition instructions.** Each instruction has the form
   $\{$state$\}$$\{$read-symbol$\}$$\{$next-state$\}$$\{$written-symbol$\}$$\{$move$\}$
   The instructions map each (state,symbol) pair to a (state,symbol,move) triple and thus
   form a transition function $\delta : Q \times Σ \mapsto Q \times Σ \times \{L, R, S\}$.

**States.** $\{q_0, q_1, A, E\}$

**Symbols.** $\{$*, 0, 1, UnderTest, #$.\}$

**Machine-level transition instructions.**

\begin{align*}
&\{q_0\}\{\ast\}\{q_0\}\{\ast\}\{R\} \\
&\{q_0\}\{1\}\{q_0\}\{1\}\{R\} \\
&\{q_0\}\{0\}\{q_1\}\{0\}\{R\} \\
&\{q_0\}\{\#\}\{E\}\{\#\}\{S\} \\
&\{q_0\}\{\text{\_}\}\{E\}\{\text{\_}\}\{S\} \\
&\{q_1\}\{1\}\{q_0\}\{1\}\{L\} \\
&\{q_1\}\{0\}\{A\}\{0\}\{S\} \\
&\{q_1\}\{\#\}\{A\}\{\#\}\{S\} \\
&\{q_1\}\{\text{\_}\}\{A\}\{\text{\_}\}\{S\}
\end{align*}

The description of a Turing Machine is a finite binary string.
Turing machines are countable and can be listed: $\{M_1, M_2, \ldots\}$.
The problems solvable by an algorithm are countable: $\{L(M_1), L(M_2), \ldots\}$. 

Creator: Malik Magdon-Ismail

Turing Machines: 13 / 13
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
Last Time: Turing Machines

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

\[ \mathcal{L} = \{ \langle G \rangle \mid G \text{ is connected} \} \]
Intuitive notion of algorithm $\equiv$ Turing Machine
Solvable problem $\equiv$ Turing-decidable

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

$$\langle G \rangle$$

($$\langle G \rangle$$ 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

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

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

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

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

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

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

(\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.)
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 ≡ Turing Machine
Solvable problem ≡ Turing-decidable

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

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

<table>
<thead>
<tr>
<th>Intuitive notion of algorithm</th>
<th>≡</th>
<th>Turing Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solvable problem</td>
<td>≡</td>
<td>Turing-decidable</td>
</tr>
</tbody>
</table>

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

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

(\( \langle G \rangle \) 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**.
Programmable Turing Machine: Universal Turing Machine

Creator: Malik Magdon-Ismail

Unsolvable Problems: 4 / 13
A Turing Machine $M$ has a binary encoding $\langle M \rangle$. 
A Turing Machine $M$ has a binary encoding $\langle M \rangle$. Its input $w$ is a binary string.
A Turing Machine $M$ has a binary encoding $\langle M \rangle$. Its input $w$ is a binary string.

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

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

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

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

Creator: Malik Magdon-Ismail
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 $U_{\text{TM}}$.

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

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

---

Creator: Malik Magdon-Ismail
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 $U_{TM}$.

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

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

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

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

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

**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$\sum$
Post’s Correspondence Problem (PCP) and **HALFSUM**

**PCP:** Consider 3 dominos:

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

**PCP:** Consider 3 dominos:

<table>
<thead>
<tr>
<th></th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>01</td>
<td>110</td>
</tr>
<tr>
<td>100</td>
<td>00</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

\[ d_3 = \begin{array}{c} 1 \newline 10 \newline 11 \end{array} \]
### 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>0</td>
<td>01</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>00</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

\[
d₃d₂ = \begin{array}{cc}
110 & 01 \\
11 & 00 \\
\end{array}
\]
Post’s Correspondence Problem (PCP) and \textbf{HALFSUM}

**PCP:** Consider 3 dominos:

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

\[
d_3d_2d_3 = \begin{array}{c}
110 \\
110 \\
11
\end{array}
\begin{array}{c}
01 \\
00 \\
11
\end{array}
\begin{array}{c}
110 \\
11 \\
01 \\
00 \\
110 \\
11
\end{array}
\]

Creator: Malik Magdon-Ismail
Unsolvable Problems: 5 / 13
**Post’s Correspondence Problem (PCP) and HALFSUM**

**PCP**: Consider 3 dominos: 

\[ d_1 \quad d_2 \quad d_3 \]

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

\[ d_3d_2d_3d_1 = \begin{array}{c c c c}
110 & 01 & 110 & 0 \\
11 & 00 & 11 & 100
\end{array} \]
PCP: Consider 3 dominos:

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

\[
d_3d_2d_3d_1 = \begin{array}{c|c|c|c}
110 & 01 & 110 & 0 \\
11 & 00 & 11 & 100 \\
\end{array} = \begin{array}{c}
110011001100110011001100 \\
\end{array}
\]
Post’s Correspondence Problem (PCP) and **HALFSUM**

**PCP:** Consider 3 dominos:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>01</th>
<th>110</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>00</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

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

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

**PCP:** Consider 3 dominos: \[ d_1 \quad d_2 \quad d_3 \]

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

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

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

**INPUT:** Dominos \( \{d_1, d_2, \ldots, d_n\} \). For example \( \left\{ \begin{array}{c|c|c}
10 & 011 & 101 \\
101 & 11 & 011 \\
\end{array} \right\} \).
**Post’s Correspondence Problem (PCP) and HALFSUM**

**PCP:** Consider 3 dominos: $d_1, d_2, d_3$

<table>
<thead>
<tr>
<th></th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>01</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>00</td>
<td>11</td>
</tr>
</tbody>
</table>

$d_3d_2d_3d_1 = \begin{bmatrix} 110 & 01 & 110 & 0 \\ 11 & 00 & 11 & 100 \end{bmatrix} = \begin{bmatrix} 110011100 \\ 110011100 \end{bmatrix}$ ← Top and bottom strings match. That’s the goal.

**INPUT:** Dominos \{d_1, d_2, \ldots, d_n\}. For example \{\begin{bmatrix} 10 \\ 101 \end{bmatrix}, \begin{bmatrix} 011 \\ 11 \end{bmatrix}, \begin{bmatrix} 101 \\ 011 \end{bmatrix}\}.

**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: 

\[
\begin{array}{c|c|c|}
\text{d}_1 & \text{d}_2 & \text{d}_3 \\
\hline
0 & 1 & 110 \\
100 & 01 & 11 \\
\end{array}
\]

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

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

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

**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 \textbf{HALFSum}

\textbf{PCP}: Consider 3 dominos: 
\[
\begin{array}{ccc}
d_1 & d_2 & d_3 \\
0 & 01 & 110 \\
100 & 00 & 11
\end{array}
\]

\[
d_3d_2d_3d_1 = \begin{array}{cccc}
110 & 01 & 110 & 0 \\
11 & 00 & 11 & 100
\end{array} = \begin{array}{c}
11001100 \\
110011100
\end{array}
\]

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

\textbf{INPUT}: Dominos \{d_1, d_2, \ldots, d_n\}. For example \[
\begin{array}{ccc}
10 & 011 & 101 \\
101 & 11 & 011
\end{array}
\].

\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, \(S = \{1, 1, 1, 3, 4, 4, 5, 6, 9\}\).
Post's Correspondence Problem (PCP) and \textbf{HALFSum}

**PCP:** Consider 3 dominos:

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

\[
d_3d_2d_3d_1 = \begin{array}{c|c|c|c|c|c|c|c|c}
& 110 & 01 & 110 & 0 & 1101100 \\
\hline
11 & 00 & 11 & 100 & & 11001100 \\
\end{array}
\]

\[\text{Top and bottom strings match. That's the goal.}\]

**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?

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\( U_{\text{TM}} \) 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 \( A_{\text{TM}} \):

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

**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

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

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

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

- A decider must \emph{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 \( A_{\text{TM}} \):

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

\textbf{input:} \( \langle M \rangle \) where \( M \) is a Turing Machine.

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

\( D_{\text{TM}} \) does the \emph{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 ($L_{TM}$ 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>$\ldots$</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>$\langle D_{TM} \rangle$</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></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></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 \( (L_{TM} \text{ Cannot be Solved}) \)

\( A_{TM} \) 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 \( A_{TM}(\langle M_i \rangle \# \langle M_j \rangle) \).

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

\( D_{TM}(\langle M_i \rangle) \) 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>$\cdots$</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>$\langle D_{TM} \rangle$</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>

$D_{TM}(\langle M_i \rangle)$ 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 $A_{TM}(\langle M_i \rangle \# \langle M_j \rangle)$.

\[
\begin{array}{l|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_{TM} \rangle & \ldots \\
\hline
\langle M_1 \rangle & \text{ACCEPT} & \text{ACCEPT} & \text{REJECT} & \text{ACCEPT} & \text{ACCEPT} & \ldots \\
\langle M_2 \rangle & \text{REJECT} & \text{REJECT} & \text{REJECT} & \text{ACCEPT} & \text{ACCEPT} & \ldots \\
\langle M_3 \rangle & & & & & & \\
\langle M_4 \rangle & & & & & & \\
\langle D_{TM} \rangle & \text{REJECT} & \text{ACCEPT} & & & & \\
\vdots & & & & & &
\end{array}
\]

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

Creator: Malik Magdon-Ismail
Theorem. \( A_{\text{TM}} \) does not exist (\( L_{\text{TM}} \) Cannot be Solved)

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

\( D_{\text{TM}} \) 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 \( A_{\text{TM}}(\langle M_i \rangle \# \langle M_j \rangle) \).

<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_{\text{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_{\text{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>

\( D_{\text{TM}}(\langle M_i \rangle) \) does the opposite of \( A_{\text{TM}}(\langle M_i \rangle \# \langle M_i \rangle) \).
Theorem. \( A_{TM} \) does not exist (\( L_{TM} \) 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>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>ACCEPT</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>( \cdots )</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>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 ($L_{TM}$ 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>ACCEPT</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 $A_{TM}(\langle M_i \rangle \# \langle M_i \rangle)$.  

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><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>$\ldots$</td>
<td></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 $A_{TM}(\langle M_i \rangle \# \langle M_i \rangle)$. 

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>$\cdots$</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>ACCEPT</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>$\cdots$</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>$\cdots$</td>
</tr>
<tr>
<td>$\langle D_{TM} \rangle$</td>
<td>ACCEPT</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>$\cdots$</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 $A_{\text{TM}}(\langle M_i \rangle \# \langle M_j \rangle)$.

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

$D_{\text{TM}}(\langle M_i \rangle)$ does the *opposite* of $A_{\text{TM}}(\langle M_i \rangle \# \langle M_i \rangle)$.
**Theorem.** \( A_{TM} \) does not exist (\( L_{TM} \) 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>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>ACCEPT?</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
</tbody>
</table>

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

DFA
(no external memory)
(regular expressions)
\{\ast01\ast\}, \{0\ast^{3n+1}\}

Creator: Malik Magdon-Ismail
Unsolvable Problems: 11 / 13
The Path Forward →
The Landscape

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

CFG
(stack)
\{0•n1•n\},
\{wwR\}

Creator: Malik Magdon-Ismail
Unsolvable Problems: 11 / 13
The Path Forward →
The Landscape

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

- **CFG** (stack)
  - \{0•n1•n\}, \{ww\}

- **TM-Decider** (RAM)
  - \{ww\}, \{02n\}, \{0•n1•n0•n\}

**HalfSum**

Creator: Malik Magdon-Ismail
Unsolvable Problems: 11 / 13
The Path Forward →
The Landscape

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

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

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

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

The Path Forward
The Landscape

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

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

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

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

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

FOCS
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

Unsolvable Problems: 12 / 13
The Path Forward: Focus on Decidable Problems

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

CFG Parsing

DFA RegExp

FOCS

Theory of Computing

Discrete Math
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

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

FOCS

Theory of Computing

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

CFG
Parsing

DFA
RegExp

Discrete Math

Proof, logic
INDUCTION

Recursion
Struct. Induction

Sums, Asymptotics
Number theory

Graph theory
Linear Algebra
Probability Theory
Multivariate Calc.

Graphs
Counting
Probability

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

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

Graph theory
Linear Algebra
Probability Theory
Multivariate Calc.

FAST (P)
Polynomial

Unsolvable Problems: 12 / 13
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

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

Decider

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

$TM = \text{Algorithm}$

Theory of Computing

CFG

Parsing

DFA

RegExp

Proof, logic

INDUCTION

Recursion

Struct. Induction

Sums, Asymptotics

Number theory

Graphs

Counting

Probability

FAST ($P$)

Polynomial

FAST ($NP$)

Unbounded Parallelism

SLOW

Exponential

FOCS

Discrete Math

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

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

SLOW
Exponential

FAST (P)
Polynomial

FAST (NP)
Unbounded Parallelism

P = NP?

Unsolvable Problems: 12 / 13

Epic Disasters →
The Path Forward: Focus on Decidable Problems

FOCS
Theory of Computing
Decider
$U_{tm} = \text{computer}$
$TM = \text{Algorithm}$

CFG
Parsing

DFA
RegExp

Sums, Asymptotics
Number theory
Graphs
Counting
Probability

Proof, logic
INDUCTION
Recursion
Struct. Induction

Graph theory
Linear Algebra
Probability Theory
Multivariate Calc.

FAST (P)
Polynomial
FAST (NP)
Unbounded Parallelism
SLOW
Exponential
Boolean Circuits

P = NP?

Parallelism
Efficiency

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

Proof, logic

INDUCTION

Recursion
Struct. Induction

Sums, Asymptotics

Number theory

Graphs

Counting

Probability

Graph theory
Linear Algebra
Probability Theory
Multivariate Calc.

FAST (P)
Polynomial

FAST (NP)
Unbounded Parallelism

SLOW
Exponential

Boolean Circuits

P = NP?

Chapters 28 & 29

Efficiency

Computation & Complexity

Introduction to Algorithms

Unsolvable Problems: 12 / 13
Epic Disasters →
The Path Forward: Focus on Decidable Problems

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

Theory of Computing
CFG
Parsing
DFA
RegExp

Discrete Math
Graph theory
Linear Algebra
Probability Theory
Multivariate Calc.

FOCS
Proof, logic
INDUCTION

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

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

Chapters
28 & 29

P = NP?

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

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.

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

Theory
Algorithms
AI

Introduction to Algorithms

Principles of Software

Computer Organization

FOCS

DISCRETE MATH

THEORY OF COMPUTING

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.

Proof, logic
INDUCTION

Recursion
Struct. Induction

Sums, Asymptotics

Number theory

Graphs

Counting

Probability

FAST (P)
Polynomial

FAST (NP)
Unbounded
Parallelism

SLOW
Exponential

Boolean Circuits

P = NP?

Chapters
28 & 29

Computability & Complexity

Algorithms & DS
- Approximation
- Randomized
- Distributed

Cryptography

Data
- ML/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

Computer
Organization

Software
Systems

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.

- Proof, logic
  - INDUCTION

- Recursion
- Struct. Induction

- Sums, Asymptotics

- Number theory

- Graphs
- Counting
- Probability

- FAST (P)
  - Polynomial

- FAST (NP)
  - Unbounded Parallelism

- SLOW
  - Exponential

- Boolean Circuits

- Chapters 28 & 29

- Computability & Complexity
  - Algorithms & DS
    - Approximation
    - Randomized
    - Distributed
  - Cryptography
  - Data
    - ML/AI/DM/NLP
    - Vision
    - Graphics
    - Comp. Finance

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

- Programming Languages
  - Compilers
  - Distributed

- Program Analysis
  - Testing
  - Verification

- DB Systems
- Parallel computing
- Operating systems
- Architecture

- Robotics
- Security

- Principles of Software
- Computer Organization
- Software Systems

- Theory Algorithms AI

- Introduction to Algorithms

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

FOCS

Theory of Computing

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

TM = Algorithm

CFG

Parsing

DFA

RegExp

Graph theory

Linear Algebra

Probability Theory

Multivariate Calc.

Decidable Problems

Computability & Complexity

P = NP?

Fonts,

Introduction to Algorithms

Principles of Software

Theory

Algorithms

AI

FOCS

Induction

Proof, logic

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

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

$TM = \text{Algorithm}$

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

P = NP?

Chapters 28 & 29

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

Computers
Operating systems
Architecture

Theory
Algorithms

AI

Introduction
to Algorithms

Principles
of Software

Computer
Organization

Software
Systems

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).


- **Pentium floating point long-division bug (1993).** Cost: $475 million – flawed division table.

- **Ariane rocket explosion (1996).** Cost: $500 million – overflow in 64-bit to 16-bit conversion.

- **Y2K (1999).** Cost: $500 spent because year was stored as 2 digits to save space.

- **Mars Climate Orbiter Crash (1998).** Cost: $125 million lost due to metric to imperial units bug.

- **Tesla Self-Driving Car (2016). 1 dead.** Auto-pilot didn’t “see” tractor-trailer.

**Financial Disasters:** London Stock Exchange down due to single server bug (**2009**; billions of pounds of trading); Knight Capital computer glitch triggers stock sale (**2012**; 500 million lost and Knight’s value drops by 75%).

**Airline Disasters:**

- AirFrance 447 2009, **228 dead:** pitot-tube failure feeds inconsistent data to programs which then panic pilot.
- Spanair 5022, 2008, **154 dead:** malware virus.
- AdamAir 574, 2007, **102 dead:** navigation system errors (and pilot errors).
- KoreanAir 801, 1997, **228 dead:** ground proximity warning system bug.
- AeroPeru 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.
...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.
  - Spanair 5022, 2008, **154 dead**: malware virus.
  - AdamAir 574, 2007, **102 dead**: navigation system errors (and pilot errors).
  - KoreanAir 801, 1997, **228 dead**: ground proximity warning system bug.
  - AeroPerù 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× 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 trigers 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.
- Spanair 5022, 2008, **154 dead**: malware virus.
- 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.**

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