The Class RE
The Story So Far

- Regular Languages
- CFLs
- $R$
- $RE$
- $A_{TM}$
- HALT
- All Languages
The Class RE

- Languages $L$ that are in RE, but not $R$, are those where:
  - We can build a TM $M$, where $\mathcal{L}(M) = L$
  - That TM $M$ has the risk of getting stuck in an infinite loop for at least some input string(s)
    - But by definition of $\mathcal{L}(M)$, only input strings that are not in $L$ are at risk of looping in $M$
  - Just like the class Regular was defined in multiple ways (DFAs, NFAs, RegExes), today we’ll learn another way to define this class RE!
Get ready to answer some questions in rapid-fire style!
(about 10 seconds per question)
Definition:

A **k-Clique** is a set of $k$ vertices of a graph that are all adjacent to each other (all possible edges between those $k$ vertices are present in the graph).

*has* a 4-Clique: \hspace{2cm} *does not* have a 4-Clique (has a 3-Clique though):

![Graph G1](image1.png)

![Graph G2](image2.png)
QUICK REACTION: Does this graph contain a 4-clique?
Reflection:

Hm, that was kind of hard to assess in just 10 seconds! What if I select and highlight just some of the nodes for you, would that be a helpful hint?
WITH A “HINT”(?): Does this graph contain a 4-clique?
Reflection:

That was a terrible so-called “hint”! It didn’t make the problem any easier to solve. :-(
WITH A NEW HINT: Does this graph contain a 4-clique?
Reflection:

The hint format (highlight some subset of 4 nodes) was a good format, but the hint is only helpful if the contents are the correct subset.
Discussion Question:

We found an effective, concise hint format for proving that a graph has a 4-Clique.

What about for proving a graph does not have a 4-Clique? What would an effective, concise hint format for that look like?
Key intuition behind our next way of defining RE:

A language \( L \) is in \textbf{RE} if, for any string \( w \), \textbf{if} you know that \( w \in L \), then there is some piece of evidence (a “hint”) you could provide to make the problem of checking that fact very easy.
More examples of helpful hints
vs
unhelpful hints
Does this Sudoku puzzle have a solution?
Verification

Does this Sudoku puzzle have a solution?
Verification

Does this Sudoku puzzle have a solution?
Verification

Does this graph have a Hamiltonian path (a simple path that passes through every node exactly once?)
Verification

Does this graph have a **Hamiltonian path** (a simple path that passes through every node exactly once?)
Verification

Does this graph have a Hamiltonian path (a simple path that passes through every node exactly once?)
Verification

Does this Sudoku puzzle have a solution?
Verification

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

Does this Sudoku puzzle have a solution?
Does this Sudoku puzzle have a solution?
Verification

Does this graph have a Hamiltonian path (a simple path that passes through every node exactly once?)
Does this graph have a **Hamiltonian path** (a simple path that passes through every node exactly once?)
Verification

Does this graph have a Hamiltonian path (a simple path that passes through every node exactly once?)
Verification

• In each of the preceding cases, we were given some problem and some evidence supporting the claim that the answer is “yes.”

• Given correct/helpful evidence, we can quickly see that the answer is indeed “yes.”

• Given incorrect/unhelpful evidence, we aren't immediately sure whether the answer is “yes.”
  • Maybe there's no evidence saying that the answer is “yes,” because the answer is no!
  • Or maybe there is some evidence, but just not the evidence we were given.

• Let's formalize this idea.
Verifiers

- A **verifier** for a language $L$ is a TM $V$ with the following properties:
  - $V$ halts on all inputs.
  - For any string $w \in \Sigma^*$, the following is true:
    $$w \in L \iff \exists c \in \Sigma^*. V \text{ accepts } \langle w, c \rangle$$
- A string $c$ where $V$ accepts $\langle w, c \rangle$ is called a **certificate** for $w$.
  - This is the “evidence.”
- Intuitively, what does this mean?
Deciders and Verifiers

Decider $M$ for $L$

- Input string ($w$)
- If $M$ accepts, then $w \in L$
- If $M$ rejects, then $w \notin L$

Verifier $V$ for $L$

- Input string ($w$)
- Certificate ($c$)
- If $V$ accepts $(w, c)$, then $w \in L$
- If $V$ rejects $(w, c)$, we don't know whether $w \in L$
Deciders and Verifiers

**Decider** $M$ for $L$

- $M$ halts on all inputs.
- $w \in L \iff M$ accepts $w$

**Verifier** $V$ for $L$

- $V$ halts on all inputs.
- $w \in L \iff \exists c \in \Sigma^*. V$ accepts $(w, c)$

- If $M$ accepts, then $w \in L$.
- If $M$ rejects, then $w \notin L$.

- If $V$ accepts $(w, c)$, then $w \in L$.
- If $V$ rejects $(w, c)$, we don't know whether $w \in L$. 

"Solve the problem"

"Check the answer"
Deciders and Verifiers

Decider $M$ for $L$

- $M$ halts on all inputs. $w \in L \iff M$ accepts $w$

Verifier $V$ for $L$

- $V$ halts on all inputs. $w \in L \iff \exists c \in \Sigma^*. V$ accepts $(w, c)$

"Solve the problem"

- If $M$ accepts, then $w \in L$.
- If $M$ rejects, then $w \notin L$.

"Check the answer"

- If $V$ accepts $(w, c)$, then $w \in L$.
- If $V$ rejects $(w, c)$, we don't know whether $w \in L$. 
- If $V$ rejects $(w, c)$, we don't know whether $w \in L$. 

Input string $(w)$
Certificate $(c)$
Deciders and Verifiers

Decider $M$ for $L$

"Solve the problem"

If $M$ accepts, then $w \in L$.
If $M$ rejects, then $w \notin L$.

Verifier $V$ for $L$

"Check the answer"

If $V$ accepts $(w, c)$, then $w \in L$.
If $V$ rejects $(w, c)$, we don't know whether $w \in L$.

$M$ halts on all inputs.
$w \in L \leftrightarrow M$ accepts $w$

$V$ halts on all inputs.
$w \in L \leftrightarrow \exists c \in \Sigma^*. V$ accepts $(w, c)$
Deciders and Verifiers

Decider $M$ for $L$

$M$ halts on all inputs.
$w \in L \iff M$ accepts $w$

Verifier $V$ for $L$

$V$ halts on all inputs.
$w \in L \iff \exists c \in \Sigma^*. V$ accepts $(w, c)$

"Solve the problem"

If $M$ accepts, then $w \in L$.
If $M$ rejects, then $w \notin L$.

"Check the answer"

If $V$ accepts $(w, c)$, then $w \in L$.
If $V$ rejects $(w, c)$, we don't know whether $w \in L$.
Verifiers

- A **verifier** for a language $L$ is a TM $V$ with the following properties:
  - $V$ halts on all inputs.
  - For any string $w \in \Sigma^*$, the following is true:
    \[ w \in L \iff \exists c \in \Sigma^*. \text{ } V \text{ accepts } \langle w, c \rangle \]

- Some notes about $V$:
  - If $V$ accepts $\langle w, c \rangle$, then we're guaranteed $w \in L$.
  - If $V$ does not accept $\langle w, c \rangle$, then either
    - $w \in L$, but you gave the wrong $c$, or
    - $w \notin L$, so no possible $c$ will work.
Verifiers

- A **verifier** for a language $L$ is a TM $V$ with the following properties:
  - $V$ halts on all inputs.
  - For any string $w \in \Sigma^*$, the following is true:
    $$w \in L \iff \exists c \in \Sigma^*. V \text{ accepts } \langle w, c \rangle$$

- More notes about $V$:
  - Notice that $c$ is existentially quantified.
  - Notice $V$ is required to halt *always* (like a decider).
Verifiers

• A **verifier** for a language $L$ is a TM $V$ with the following properties:
  
  • $V$ halts on all inputs.
  
  • For any string $w \in \Sigma^*$, the following is true:
    
    \[ w \in L \iff \exists c \in \Sigma^*. \text{ } V \text{ accepts } \langle w, c \rangle \]

• More notes about $V$:
  
  • Notice that $\mathcal{L}(V) \neq L$. *(Good question to hold on to for a second: what is $\mathcal{L}(V)$?)*
  
  • The job of $V$ is just to check certificates, not to decide membership in $L$. 
Verifiers

- A **verifier** for a language $L$ is a TM $V$ with the following properties:
  - $V$ halts on all inputs.
  - For any string $w \in \Sigma^*$, the following is true:
    $$w \in L \iff \exists c \in \Sigma^*. V \text{ accepts } \langle w, c \rangle$$

- A note about $c$:
  - Figuring out what would make a good certificate (should it be a number of steps to take, an equation-solving variable assignment, a set of graph nodes, an array of numbers to fill in a whole Sudoku board?) is custom work to do for each different language $L$. 
Some Verifiers

• Let $L$ be the following language:

$$L = \{ \langle n \rangle \mid n \in \mathbb{N} \text{ and the hailstone sequence terminates for } n \}$$

```cpp
bool checkHailstone(int n, int c) {
    for (int i = 0; i < c; i++) {
        if (n % 2 == 0) n /= 2;
        else n = 3*n + 1;
        if (n == 1) return true;
    }
    return n == 1;
}
```
Some Verifiers

Does this always halt?

$L = \{ \langle n \rangle \mid n \in \mathbb{N} \text{ and the hailstone sequence terminates for } n \}$

```cpp
bool checkHailstone(int n, int c) {
    for (int i = 0; i < c; i++) {
        if (n % 2 == 0) n /= 2;
        else n = 3*n + 1;
        if (n == 1) return true;
    }
    return n == 1;
}
```
Some Verifiers

For one given \(\langle n \rangle \in L\) (say 11), how many different values of \(c\) will work to cause the verifier to accept?

\[ L = \{ \langle n \rangle | n \in \mathbb{N} \text{ and the hailstone sequence terminates for } n \} \]

```cpp
bool checkHailstone(int n, int c) {
    for (int i = 0; i < c; i++) {
        if (n % 2 == 0) n /= 2;
        else n = 3*n + 1;
        if (n == 1) return true;
    }
    return n == 1;
}
```
Some Verifiers

Let $L$ be the following language:

$L = \{ \langle n \rangle | n \in \mathbb{N} \text{ and the hailstone sequence terminates for } n \}$

```
bool checkHailstone(int n, int c) {
    for (int i = 0; i < c; i++) {
        if (n % 2 == 0) n /= 2;
        else n = 3*n + 1;
        if (n == 1) return true;
    }
    return n == 1;
}
```

How many of these statements are true of $\mathcal{L}(V)$?

- $\mathcal{L}(V) = L$
- $\mathcal{L}(V) \subseteq L$
- $L \subseteq \mathcal{L}(V)$
Some Verifiers

• Let $L$ be the following language:

$$L = \{ \langle G \rangle \mid G \text{ is a graph and } G \text{ has a Hamiltonian path} \}$$

• (A Hamiltonian path is a simple path that visits every node in the graph.)

• Let's see how to build a verifier for $L$. 
Is there a simple path that goes through every node exactly once?
Verifier Example: Hamiltonian Path

• Let $L$ be the following language:

$\begin{align*}
L &= \{ \langle G \rangle \mid G \text{ is a graph with a Hamiltonian path} \}
\end{align*}$

```cpp
bool checkHamiltonian(Graph G, vector<Node> c) {
    if (c.size() != G.numNodes()) return false;
    if (containsDuplicate(c)) return false;

    for (size_t i = 0; i < c.size() - 1; i++) {
        if (!G.hasEdge(c[i], c[i+1])) return false;
    }
    return true;
}
```

• Do you see why $\langle G \rangle \in L$ iff there is a $c$ where
checkHamiltonian($G, c$) returns true?
• Do you see why checkHamiltonian always halts?
Where We’ve Been

NFA

Regex

State Elimination

Thompson’s Algorithm
Where We’re Going Today

Verifier

Somehow build this

Recognizer

Somehow build this
Verifier for $A_{TM}$?

- Consider $A_{TM}$:
  \[ A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \} \].

- This is our standard example of an undecidable language. There’s no way, in general, to tell whether a TM $M$ will accept a string $w$.

- Although this language is undecidable, it’s an RE language, and it’s possible to build a verifier for it!
What would make a good certificate for a verifier for $A_{TM}$?

- Consider $A_{TM}$:

  $$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}.$$

- This is a *canonical* example of an undecidable language. There’s no way, in general, to tell whether a TM $M$ will accept a string $w$.

- Although this language is undecidable, it’s an RE language, and it’s possible to build a verifier for it!
A Verifier for $A_{TM}$

- Recall $A_{TM} = \{ \langle M, w \rangle \mid M$ is a TM and $M$ accepts $w \}$

```cpp
bool checkWillAccept(TM M, string w, int c) {
    set up a simulation of M running on w;
    for (int i = 0; i < c; i++) {
        simulate the next step of M running on w;
    }
    return whether M is in an accepting state;
}
```

- Do you see why $M$ accepts $w$ iff there is some $c$ such that checkWillAccept($M$, $w$, $c$) returns true?
- Do you see why checkWillAccept always halts?
Equivalence of Verifiers and Recognizers

Verifier

Recognizer

Enforce a step count
What languages are verifiable?
Let $V$ be a verifier for a language $L$. Consider the following function given in pseudocode:

```cpp
bool mysteryFunction(string w) {
    int i = 0;
    while (true) {
        for (each string c of length i) {
            if (V accepts $\langle w, c \rangle$) return true;
        }
        i++;
    }
}
```

What set of strings does `mysteryFunction` return `true` on?
Equivalence of Verifiers and Recognizers

- Try all certificates
- Enforce a step count
Theorem: If $L$ is a language, then there is a verifier for $L$ if and only if $L \in \text{RE}$. 
Verifiers and RE

- **Theorem**: If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.

- **Proof goal**: Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$. 
Verifiers and \textbf{RE}

- \textbf{Theorem:} If there is a verifier $V$ for a language $L$, then $L \in \textbf{RE}$.

- \textbf{Proof goal:} Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$. 

\begin{center}
\begin{tikzpicture}
  \node (V) [draw, fill=blue!20, rounded corners] {	extit{Verifier V for }$L$};
  \draw[->] (V) -- node[above] {\textit{input string (w)}} ++(1,0);
  \draw[->] (V) -- node[below] {\textit{certificate (c)}} ++(0,-1);
  \node (yes) [circle, draw, fill=green!20, above right of=V] {	extit{yes!}};
  \node (not sure) [circle, draw, fill=orange!20, below right of=V] {	extit{not sure}};
  \draw[->] (V) -- node[draw,fill=white, font={\scriptsize\textit{"Check the answer}}} {} (yes);
  \draw[->] (V) -- (not sure);
\end{tikzpicture}
\end{center}
Verifiers and \textbf{RE}

- **Theorem:** If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.

- **Proof goal:** Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)

```
ε  a  b  aa  ab  ba  bb  aaa  aab  aba  abb  baa  ...
```
Verifiers and \textbf{RE}

- **Theorem**: If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.
- **Proof goal**: Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)
Verifiers and \textbf{RE}

\textbf{Theorem}: If there is a verifier $V$ for a language $L$, then $L \in \textbf{RE}$.

\textbf{Proof goal}: Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)
Verifiers and **RE**

- **Theorem**: If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.

- **Proof goal**: Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)
Theorem: If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.

Proof goal: Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)

- $\epsilon$
- $a$
- $b$
- $aa$
- $ab$
- $ba$
- $bb$
- $aaa$
- $aab$
- $aba$
- $abb$
- $baa$
- ...
Verifiers and \textbf{RE}

- **Theorem:** If there is a verifier \( V \) for a language \( L \), then \( L \in \text{RE} \).

- **Proof goal:** Given a verifier \( V \) for a language \( L \), find a way to construct a recognizer \( M \) for \( L \).

We will try all possible certificates (values of \( c \))
Verifiers and \textbf{RE}

- **Theorem**: If there is a verifier \( V \) for a language \( L \), then \( L \in \text{RE} \).

- **Proof goal**: Given a verifier \( V \) for a language \( L \), find a way to construct a recognizer \( M \) for \( L \).

We will try all possible certificates (values of \( c \))
**Theorem:** If there is a verifier $V$ for a language $L$, then $L \in \textbf{RE}$.

**Proof goal:** Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)
Verifiers and RE

- **Theorem**: If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.

- **Proof goal**: Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)

\[ \varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, \ldots \]
Theorem: If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.

Proof goal: Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)
Verifiers and \textbf{RE}

- **Theorem**: If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.
- **Proof goal**: Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$. 

![Diagram of a verifier V for L]

- $\varepsilon$, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, ...
Verifiers and $\textbf{RE}$

- **Theorem**: If there is a verifier $V$ for a language $L$, then $L \in \textbf{RE}$.

- **Proof goal**: Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)

\[
\begin{align*}
\varepsilon & \quad a \quad b \quad aa \quad ab \quad ba \quad bb \quad aaa \quad aab \quad aba \quad abb \quad baa \quad \ldots
\end{align*}
\]
Verifiers and **RE**

- **Theorem:** If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.

- **Proof goal:** Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)

\[
\epsilon \quad a \quad b \quad aa \quad ab \quad ba \quad bb \quad aaa \quad aab \quad aba \quad abb \quad baa \quad \ldots
\]
**Theorem:** If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.

**Proof goal:** Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)

\[ \varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, \ldots \]
Verifiers and RE

- **Theorem**: If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.
- **Proof goal**: Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)

| $\varepsilon$ | a | b | aa | ab | ba | bb | aaa | aab | aba | abb | baa | ...
|---------------|---|---|----|----|----|----|-----|-----|-----|-----|-----|-----|

"Check the answer"
Verifiers and \textbf{RE}

- **Theorem:** If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.
- **Proof goal:** Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)
Verifiers and \textbf{RE}

- **Theorem:** If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.

- **Proof goal:** Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)
Verifiers and \textbf{RE}

- **Theorem**: If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.

- **Proof goal**: Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)
Verifiers and \textbf{RE}

- **Theorem:** If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.

- **Proof goal:** Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)

\[
\begin{array}{c}
\varepsilon & a & b & \text{aa} & \text{ab} & \text{ba} & \text{bb} & \text{aaa} & \text{aab} & \text{aba} & \text{abb} & \text{baa} & \ldots \\
\end{array}
\]
Verifiers and \textbf{RE}

\begin{itemize}
  \item \textbf{Theorem:} If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.
  \item \textbf{Proof goal:} Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.
\end{itemize}

We will try all possible certificates (values of $c$)
Verifiers and \textbf{RE}

- \textbf{Theorem:} If there is a verifier \( V \) for a language \( L \), then \( L \in \textbf{RE} \).

- \textbf{Proof goal:} Given a verifier \( V \) for a language \( L \), find a way to construct a recognizer \( M \) for \( L \).

\[ \varepsilon \ a \ b \ aa \ ab \ ba \ bb \ aaa \ aab \ aba \ abb \ baa \ldots \]

\begin{itemize}
  \item We will try all possible certificates (values of \( c \)).
\end{itemize}
Verifiers and RE

- **Theorem:** If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.

- **Proof goal:** Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)
Verifiers and \textbf{RE}

- \textbf{Theorem:} If there is a verifier $V$ for a language $L$, then $L \in \textbf{RE}$.

- \textbf{Proof goal:} Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)
Verifiers and RE

- **Theorem:** If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.

- **Proof goal:** Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

![Diagram showing the process of using a verifier to construct a recognizer.](image-url)
Verifiers and RE

- **Theorem**: If there is a verifier $V$ for a language $L$, then $L \in \text{RE}$.

- **Proof goal**: Given a verifier $V$ for a language $L$, find a way to construct a recognizer $M$ for $L$.

We will try all possible certificates (values of $c$)
Verifiers and \textbf{RE}

\begin{itemize}
\item \textbf{Theorem:} If \( V \) is a verifier for \( L \), then \( L \in \textbf{RE} \).
\item \textbf{Proof sketch:} Consider the following program:
\begin{verbatim}
bool isInL(string w) {
    int i = 0;
    while (true) {
        for (each string c of length i) {
            if (V accepts \langle w, c \rangle) return true;
        }
        i++;
    }
}
\end{verbatim}

If \( w \in L \), there is some \( c \in \Sigma^* \) where \( V \) accepts \( \langle w, c \rangle \). The function \texttt{isInL} tries all possible strings as certificate, so it will eventually find \( c \) (or some other certificate), see \( V \) accept \( \langle w, c \rangle \), then return true. Conversely, if \texttt{isInL}(\( w \)) returns true, then there was some string \( c \) such that \( V \) accepted \( \langle w, c \rangle \), so \( w \in L \). \textbf{■}
\end{itemize}
Verifiers and RE

• **Theorem:** If \( L \in \text{RE} \), then there is a verifier for \( L \).

• **Proof goal:** Beginning with a recognizer \( M \) for the language \( L \), show how to construct a verifier \( V \) for \( L \).

• The challenges:
  - A recognizer \( M \) is not required to halt on all inputs. A verifier \( V \) must always halt.
  - A recognizer \( M \) takes in one single input. A verifier \( V \) takes in two inputs.
  - We’ll need to find a way of reconciling these requirements.
Recall: If $M$ is a recognizer for a language $L$, then $M$ accepts $w$ iff $w \in L$.

Key insight: If $M$ accepts a string $w$, it always does so in a finite number of steps.

Idea: Adapt the verifier for $A_{\text{TM}}$ into a more general construction that turns any recognizer into a verifier by running it for a fixed number of steps.
Verifiers and RE

- **Theorem:** If \( L \in \text{RE} \), then there is a verifier for \( L \).
- **Proof sketch:** Consider the following program:

```cpp
bool checkIsInL(string w, int c) {
    set up a simulation of M running on w;
    for (int i = 0; i < c; i++) {
        simulate the next step of M running on W;
    }
    return whether M is in an accepting state;
}
```

Notice that \( \text{checkIsInL} \) always halts, since each step takes only finite time to complete. Next, notice that if there is a \( c \) where \( \text{checkIsInL}(w, c) \) returns true, then \( M \) accepted \( w \) after running for \( c \) steps, so \( w \in L \). Conversely, if \( w \in L \), then \( M \) accepts \( w \) after some number of steps (call that number \( c \)). Then \( \text{checkIsInL}(w, c) \) will run \( M \) on \( w \) for \( c \) steps, watch \( M \) accept \( w \), then return true. ■
RE and Proofs

- Verifiers and recognizers give two different perspectives on the “proof” intuition for RE.
- Verifiers are explicitly built to check proofs that strings are in the language.
  - If you know that some string \( w \) belongs to the language and you have the proof of it, you can convince someone else that \( w \in L \).
- You can think of a recognizer as a device that “searches” for a proof that \( w \in L \).
  - If it finds it, great!
  - If not, it might loop forever.
**RE and Proofs**

- If the RE languages represent languages where membership can be proven, what does a non-RE language look like?

- Intuitively, a language is *not* in RE if there is no general way to prove that a given string $w \in L$ actually belongs to $L$.

- In other words, even if you knew that a string was in the language, you may never be able to convince anyone of it!
Unsolvable Problems
Finding Non-RE Languages
Finding Non-RE Languages

- Remember RE but non-R (undecidable) languages are those where we can reliably identify strings in the language, but cannot readily identify strings that are not in the language.
- Non-RE languages will be those where we cannot even readily identify strings that are in the language!
- How might we find an example of a non-RE language?
Languages, TMs, and TM Encodings

• What happens if we list off all Turing machines, looking at how those TMs behave when given other TM codes (as strings, so various $\langle M_x \rangle$ strings) as input?
All Turing machines, listed in some order.
\[\langle M_0 \rangle \langle M_1 \rangle \langle M_2 \rangle \langle M_3 \rangle \langle M_4 \rangle \langle M_5 \rangle \ldots\]
All files/strings of TMs’ code, listed in the same order.
<table>
<thead>
<tr>
<th>M_0</th>
<th>M_1</th>
<th>M_2</th>
<th>M_3</th>
<th>M_4</th>
<th>M_5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>M_0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M_4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M_5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\langle M_0 \rangle$</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>$\langle M_5 \rangle$</td>
<td>...</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>-----</td>
</tr>
<tr>
<td>M₀</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>M₁</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>M₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>M₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>M₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>M₅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>\langle M_0 \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 M_5 \rangle</td>
<td>\ldots</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------</td>
<td>-----------------</td>
<td>-----------------</td>
<td>-----------------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td><strong>M_0</strong></td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td><strong>M_1</strong></td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td><strong>M_2</strong></td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td><strong>M_3</strong></td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td><strong>M_4</strong></td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td><strong>M_5</strong></td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td><strong>\ldots</strong></td>
<td><strong>\ldots</strong></td>
<td><strong>\ldots</strong></td>
<td><strong>\ldots</strong></td>
<td><strong>\ldots</strong></td>
<td><strong>\ldots</strong></td>
<td><strong>\ldots</strong></td>
</tr>
</tbody>
</table>
What is $L(M_0)$?

A. $\Sigma^*$
B. $\{\langle M_0 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \ldots \}$
C. $\{\langle M_0 \rangle\}$
D. $\{\langle M_0 \rangle, \langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \ldots \}$
E. Something else.
\[ L(M_0) = \{ \langle M_0 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \ldots \}. \]

And we can’t see the rest of the table for \( M_2 \), but it accepts everything so far, so it’s at least possible that its language is \( L(M_2) = \Sigma^* \).
Aside: we aren’t really worrying about the existence of other strings that aren’t TM code right now, but you could also think of it including those strings, so
\[ \mathcal{L}(M_0) = \{ \langle M_0 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \ldots \} \cup \{ w | w \text{ is a string that isn’t a TM’s code, and } M_0 \text{ accepts } w \} \]
<table>
<thead>
<tr>
<th></th>
<th>$M_0$</th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$M_4$</th>
<th>$M_5$</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_0$</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>…</td>
</tr>
<tr>
<td>$M_1$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>…</td>
</tr>
<tr>
<td>$M_2$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>…</td>
</tr>
<tr>
<td>$M_3$</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>…</td>
</tr>
<tr>
<td>$M_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_5$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_0$</td>
<td>$M_1$</td>
<td>$M_2$</td>
<td>$M_3$</td>
<td>$M_4$</td>
<td>$M_5$</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\langle M_0 \rangle$</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>$\langle M_5 \rangle$</td>
<td>...</td>
</tr>
<tr>
<td>---</td>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>-----</td>
</tr>
<tr>
<td>$M_0$</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>$M_1$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>$M_2$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>$M_3$</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>$M_4$</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>$M_5$</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>$M_0$</td>
<td>$M_1$</td>
<td>$M_2$</td>
<td>$M_3$</td>
<td>$M_4$</td>
<td>$M_5$</td>
<td>...</td>
</tr>
<tr>
<td>---</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>$M_0$</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>$M_1$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>$M_2$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>$M_3$</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>$M_4$</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>$M_5$</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Quick check:** How many of the TMs on this chart so far do **NOT** accept their own code as a string? (Enter a number.)

Answer at PollEv.com/cs103 or text **CS103** to **22333** once to join, then **a number**.
<table>
<thead>
<tr>
<th>\langle M_0 \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 M_5 \rangle</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_0</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td>M_1</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>M_2</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>M_3</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>M_4</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td>M_5</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(M_0)</td>
<td>(M_1)</td>
<td>(M_2)</td>
<td>(M_3)</td>
<td>(M_4)</td>
<td>(M_5)</td>
<td>...</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>-----</td>
</tr>
<tr>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

What are we going to do next?

Answer at PollEv.com/cs103 or text CS103 to 22333 once to join, then your answer.
<table>
<thead>
<tr>
<th>$\langle M_0 \rangle$</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>$\langle M_5 \rangle$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>$\langle M_0 \rangle$</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>$\langle M_5 \rangle$</td>
</tr>
<tr>
<td>---</td>
<td>------------------</td>
<td>------------------</td>
<td>------------------</td>
<td>------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>$M_0$</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td>$M_1$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>$M_2$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>$M_3$</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>$M_4$</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td>$M_5$</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

| Acc | Acc | Acc | No | Acc | No | … |
Flip all “accept” to “no” and vice-versa.
<table>
<thead>
<tr>
<th>$\langle M_0 \rangle$</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>$\langle M_5 \rangle$</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>M_0</strong></td>
<td><strong>Acc</strong></td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td><strong>M_1</strong></td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td><strong>M_2</strong></td>
<td>Acc</td>
<td>Acc</td>
<td><strong>Acc</strong></td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td><strong>M_3</strong></td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td><strong>M_4</strong></td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
</tr>
<tr>
<td><strong>M_5</strong></td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

<p>| No              | No              | No              | Acc             | No              | Acc             | … |</p>
<table>
<thead>
<tr>
<th></th>
<th>(M_0)</th>
<th>(M_1)</th>
<th>(M_2)</th>
<th>(M_3)</th>
<th>(M_4)</th>
<th>(M_5)</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M_0)</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>…</td>
</tr>
<tr>
<td>(M_1)</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>…</td>
</tr>
<tr>
<td>(M_2)</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>…</td>
</tr>
<tr>
<td>(M_3)</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>…</td>
</tr>
<tr>
<td>(M_4)</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>…</td>
</tr>
<tr>
<td>(M_5)</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

What TM has this behavior?
<table>
<thead>
<tr>
<th>M_0</th>
<th>M_1</th>
<th>M_2</th>
<th>M_3</th>
<th>M_4</th>
<th>M_5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>$\langle M_0 \rangle$</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>$\langle M_5 \rangle$</td>
</tr>
<tr>
<td>---</td>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>$M_0$</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td>$M_1$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>$M_2$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>$M_3$</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>$M_4$</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
</tr>
<tr>
<td>$M_5$</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<p>| No | No | No | Acc | No | Acc | ... |</p>
<table>
<thead>
<tr>
<th>( \langle M_0 \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 M_5 \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>M_0</strong></td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td><strong>M_1</strong></td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td><strong>M_2</strong></td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td><strong>M_3</strong></td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
</tr>
<tr>
<td><strong>M_4</strong></td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
</tr>
<tr>
<td><strong>M_5</strong></td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td><strong>...</strong></td>
<td><strong>...</strong></td>
<td><strong>...</strong></td>
<td><strong>...</strong></td>
<td><strong>...</strong></td>
<td><strong>...</strong></td>
</tr>
<tr>
<td><strong>...</strong></td>
<td><strong>No</strong></td>
<td><strong>No</strong></td>
<td><strong>No</strong></td>
<td><strong>Acc</strong></td>
<td><strong>No</strong></td>
</tr>
<tr>
<td></td>
<td>$M_0$</td>
<td>$M_1$</td>
<td>$M_2$</td>
<td>$M_3$</td>
<td>$M_4$</td>
</tr>
<tr>
<td>---</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>$M_0$</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>$M_1$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>$M_2$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>$M_3$</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
</tr>
<tr>
<td>$M_4$</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
</tr>
<tr>
<td>$M_5$</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
</tr>
<tr>
<td>M_0</td>
<td>M_1</td>
<td>M_2</td>
<td>M_3</td>
<td>M_4</td>
<td>M_5</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<p>| No  | No  | No  | Acc | No  | Acc | ... |</p>
<table>
<thead>
<tr>
<th>M_0</th>
<th>M_1</th>
<th>M_2</th>
<th>M_3</th>
<th>M_4</th>
<th>M_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

No No No Acc No Acc …
<table>
<thead>
<tr>
<th>\langle M_0 \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 M_5 \rangle</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>M_0</strong></td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td><strong>M_1</strong></td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td><strong>M_2</strong></td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td><strong>M_3</strong></td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td><strong>M_4</strong></td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td><strong>M_5</strong></td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>(\langle M_0 \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 M_5 \rangle)</td>
</tr>
<tr>
<td>---</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>(M_0)</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td>(M_1)</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>(M_2)</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>(M_3)</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
</tr>
<tr>
<td>(M_4)</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
</tr>
<tr>
<td>(M_5)</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

<p>| No | No | No | Acc | No | Acc | … |</p>
<table>
<thead>
<tr>
<th></th>
<th>$M_0$</th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$M_4$</th>
<th>$M_5$</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_0$</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>…</td>
</tr>
<tr>
<td>$M_1$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>…</td>
</tr>
<tr>
<td>$M_2$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>…</td>
</tr>
<tr>
<td>$M_3$</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>…</td>
</tr>
<tr>
<td>$M_4$</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>…</td>
</tr>
<tr>
<td>$M_5$</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

<p>| No | No | No | Acc | No | Acc | … |</p>
<table>
<thead>
<tr>
<th></th>
<th>$M_0$</th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$M_4$</th>
<th>$M_5$</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_0$</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>…</td>
</tr>
<tr>
<td>$M_1$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>…</td>
</tr>
<tr>
<td>$M_2$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>…</td>
</tr>
<tr>
<td>$M_3$</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>…</td>
</tr>
<tr>
<td>$M_4$</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>…</td>
</tr>
<tr>
<td>$M_5$</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

No No No Acc No Acc …
<table>
<thead>
<tr>
<th>M_0</th>
<th>M_1</th>
<th>M_2</th>
<th>M_3</th>
<th>M_4</th>
<th>M_5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

No TM has this behavior!
<table>
<thead>
<tr>
<th></th>
<th>(M_0)</th>
<th>(M_1)</th>
<th>(M_2)</th>
<th>(M_3)</th>
<th>(M_4)</th>
<th>(M_5)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(M_0)</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>(M_1)</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td></td>
</tr>
<tr>
<td>(M_2)</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td></td>
</tr>
<tr>
<td>(M_3)</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td></td>
</tr>
<tr>
<td>(M_4)</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>(M_5)</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

|   | No      | No      | No      | Acc     | No      | Acc     |   |

Note: The table contains entries indicating whether certain conditions are met (Acc) or not (No). The entries are organized in a way that suggests a pattern or sequence across the columns.
<table>
<thead>
<tr>
<th></th>
<th>$M_0$</th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$M_4$</th>
<th>$M_5$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_0$</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>$M_1$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>$M_2$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>$M_3$</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>$M_4$</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>$M_5$</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>...</td>
</tr>
</tbody>
</table>

... ... ... ... ... ... ... ...

No No No Acc No Acc ...
<table>
<thead>
<tr>
<th></th>
<th>$M_0$</th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$M_4$</th>
<th>$M_5$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_0$</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>$M_1$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>$M_2$</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>$M_3$</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>...</td>
</tr>
<tr>
<td>$M_4$</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>Acc</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>$M_5$</td>
<td>No</td>
<td>No</td>
<td>Acc</td>
<td>Acc</td>
<td>No</td>
<td>No</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

“The language of all TMs that do not accept their own description.”
$$\{ \langle M \rangle | M \text{ is a TM that does not accept } \langle M \rangle \}$$
\[
\{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin L(M) \} 
\]
Diagonalization Revisited

• The *diagonalization language*, which we denote $L_D$, is defined as

$$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathcal{L}(M) \}$$

• That is, $L_D$ is the set of descriptions of Turing machines that do not accept themselves.
\[ L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathcal{L}(M) \} \]

**Theorem:** \( L_D \notin \text{RE}. \)
\[ L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathcal{L}(M) \} \]

**Theorem:** \( L_D \notin \text{RE} \).

**Proof:** By contradiction; assume that \( L_D \in \text{RE} \).

From the definition of \( L_D \), we see that \( \langle M \rangle \in L_D \) iff \( \langle M \rangle \notin \mathcal{L}(M) \).

Combining this with statement (1) tells us that \( \langle M \rangle \notin \mathcal{L}(M) \) iff \( \langle M \rangle \in \mathcal{L}(R) \) (2).

Statement (2) holds for any TM \( M \), so in particular it should hold when \( M = R \). If we pick \( M = R \), we see that \( \langle R \rangle \notin \mathcal{L}(R) \) iff \( \langle R \rangle \in \mathcal{L}(R) \) (3).

This is clearly impossible. We have reached a contradiction, so our assumption must have been wrong. Thus \( L_D \notin \text{RE} \). \( \blacksquare \)
\[ L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathcal{L}(M) \} \]

**Theorem:** \( L_D \notin \text{RE}. \)

**Proof:** By contradiction; assume that \( L_D \in \text{RE}. \) Then there must be some recognizer \( R \) such that \( \mathcal{L}(R) = L_D. \)
\[ L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathcal{L}(M) \} \]

**Theorem:** \( L_D \notin \text{RE} \).

**Proof:** By contradiction; assume that \( L_D \in \text{RE} \). Then there must be some recognizer \( R \) such that \( \mathcal{L}(R) = L_D \).

Let \( M \) be an arbitrary TM. Since \( \mathcal{L}(R) = L_D \), we know that
\[
\langle M \rangle \in L_D \iff \langle M \rangle \in \mathcal{L}(R). \quad (1)
\]
\[ L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathcal{L}(M) \} \]

**Theorem:** \( L_D \notin \text{RE}. \)

**Proof:** By contradiction; assume that \( L_D \in \text{RE}. \) Then there must be some recognizer \( R \) such that \( \mathcal{L}(R) = L_D. \)

Let \( M \) be an arbitrary TM. Since \( \mathcal{L}(R) = L_D \), we know that

\[
\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathcal{L}(R). \quad (1)
\]

Because \( \mathcal{L}(R) = L_D \), we know that a string belongs to one set if and only if it belongs to the other.
\( L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathcal{L}(M) \} \)

**Theorem:** \( L_D \notin \text{RE} \).

**Proof:** By contradiction; assume that \( L_D \in \text{RE} \). Then there must be some recognizer \( R \) such that \( \mathcal{L}(R) = L_D \).

Let \( M \) be an arbitrary TM. Since \( \mathcal{L}(R) = L_D \), we know that

\[
\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathcal{L}(R). \tag{1}
\]

From the definition of \( L_D \), we see that \( \langle M \rangle \in L_D \) iff \( \langle M \rangle \notin \mathcal{L}(M) \).
\[ L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathcal{L}(M) \} \]

**Theorem:** \( L_D \notin \text{RE} \).

**Proof:** By contradiction; assume that \( L_D \in \text{RE} \). Then there must be some recognizer \( R \) such that \( \mathcal{L}(R) = L_D \).

Let \( M \) be an arbitrary TM. Since \( \mathcal{L}(R) = L_D \), we know that

\[
\langle M \rangle \in L_D \iff \langle M \rangle \in \mathcal{L}(R). \quad (1)
\]

From the definition of \( L_D \), we see that \( \langle M \rangle \in L_D \) iff \( \langle M \rangle \notin \mathcal{L}(M) \).

Combining this with statement (1) tells us that

\[
\langle M \rangle \notin \mathcal{L}(M) \iff \langle M \rangle \in \mathcal{L}(R). \quad (2)
\]

This is clearly impossible. We have reached a contradiction, so our assumption must have been wrong. Thus \( L_D \notin \text{RE} \). \( \blacksquare \)
\[ L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathcal{L}(M) \} \]

**Theorem:** \( L_D \notin \text{RE}. \)

**Proof:** By contradiction; assume that \( L_D \in \text{RE}. \) Then there must be some recognizer \( R \) such that \( \mathcal{L}(R) = L_D. \)

Let \( M \) be an arbitrary TM. Since \( \mathcal{L}(R) = L_D, \) we know that

\[
\langle M \rangle \in L_D \iff \langle M \rangle \in \mathcal{L}(R). \quad (1)
\]

From the definition of \( L_D, \) we see that \( \langle M \rangle \in L_D \iff \langle M \rangle \notin \mathcal{L}(M). \)

Combining this with statement (1) tells us that

\[
\langle M \rangle \notin \mathcal{L}(M) \iff \langle M \rangle \in \mathcal{L}(R). \quad (2)
\]

We've replaced the left-hand side of this biconditional with an equivalent statement.
Theorem: $L_D \notin \text{RE}$.

Proof: By contradiction; assume that $L_D \in \text{RE}$. Then there must be some recognizer $R$ such that $\mathcal{L}(R) = L_D$.

Let $M$ be an arbitrary TM. Since $\mathcal{L}(R) = L_D$, we know that

\[
\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathcal{L}(R). \quad (1)
\]

From the definition of $L_D$, we see that $\langle M \rangle \in L_D$ iff $\langle M \rangle \notin \mathcal{L}(M)$. Combining this with statement (1) tells us that

\[
\langle M \rangle \notin \mathcal{L}(M) \text{ iff } \langle M \rangle \in \mathcal{L}(R). \quad (2)
\]

Since our choice of $M$ was arbitrary, we see that statement (2) holds for any TM $M$. 

$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathcal{L}(M) \}$
\[ L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathcal{L}(M) \} \]

**Theorem:** \( L_D \notin \text{RE}. \)

**Proof:** By contradiction; assume that \( L_D \in \text{RE}. \) Then there must be some recognizer \( R \) such that \( \mathcal{L}(R) = L_D. \)

Let \( M \) be an arbitrary TM. Since \( \mathcal{L}(R) = L_D, \) we know that

\[ \langle M \rangle \in L_D \iff \langle M \rangle \in \mathcal{L}(R). \quad (1) \]

From the definition of \( L_D, \) we see that \( \langle M \rangle \in L_D \iff \langle M \rangle \notin \mathcal{L}(M). \)

Combining this with statement (1) tells us that

\[ \langle M \rangle \notin \mathcal{L}(M) \iff \langle M \rangle \in \mathcal{L}(R). \quad (2) \]

Since our choice of \( M \) was arbitrary, we see that statement (2) holds for any TM \( M. \)

A nice consequence of a universally-quantified statement is that it should work in all cases.
Theorem: $L_D \not\in \text{RE}$.

Proof: By contradiction; assume that $L_D \in \text{RE}$. Then there must be some recognizer $R$ such that $\mathcal{L}(R) = L_D$.

Let $M$ be an arbitrary TM. Since $\mathcal{L}(R) = L_D$, we know that

$$
\langle M \rangle \in L_D \iff \langle M \rangle \in \mathcal{L}(R).
$$

(1)

From the definition of $L_D$, we see that $\langle M \rangle \in L_D$ iff $\langle M \rangle \notin \mathcal{L}(M)$.

Combining this with statement (1) tells us that

$$
\langle M \rangle \notin \mathcal{L}(M) \iff \langle M \rangle \in \mathcal{L}(R).
$$

(2)

Since our choice of $M$ was arbitrary, we see that statement (2) holds for any TM $M$. In particular, this means that statement (2) holds for the TM $R$, which tells us that

$$
\langle R \rangle \notin \mathcal{L}(R) \iff \langle R \rangle \in \mathcal{L}(R).
$$

(3)
Theorem: $L_D \notin \text{RE}$.

Proof: By contradiction; assume that $L_D \in \text{RE}$. Then there must be some recognizer $R$ such that $\mathcal{L}(R) = L_D$.

Let $M$ be an arbitrary TM. Since $\mathcal{L}(R) = L_D$, we know that

$$\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathcal{L}(R). \quad (1)$$

From the definition of $L_D$, we see that $\langle M \rangle \in L_D \iff \langle M \rangle \notin \mathcal{L}(M)$. Combining this with statement (1) tells us that

$$\langle M \rangle \notin \mathcal{L}(M) \iff \langle M \rangle \in \mathcal{L}(R). \quad (2)$$

Since our choice of $M$ was arbitrary, we see that statement (2) holds for any TM $M$. In particular, this means that statement (2) holds for the TM $R$, which tells us that

$$\langle R \rangle \notin \mathcal{L}(R) \iff \langle R \rangle \in \mathcal{L}(R). \quad (3)$$

This is clearly impossible.
\[ L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathcal{L}(M) \} \]

**Theorem:** \( L_D \notin \mathbf{RE} \).

**Proof:** By contradiction; assume that \( L_D \in \mathbf{RE} \). Then there must be some recognizer \( R \) such that \( \mathcal{L}(R) = L_D \).

Let \( M \) be an arbitrary TM. Since \( \mathcal{L}(R) = L_D \), we know that

\[ \langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathcal{L}(R). \tag{1} \]

From the definition of \( L_D \), we see that \( \langle M \rangle \in L_D \) iff \( \langle M \rangle \notin \mathcal{L}(M) \).

Combining this with statement (1) tells us that

\[ \langle M \rangle \notin \mathcal{L}(M) \text{ iff } \langle M \rangle \in \mathcal{L}(R). \tag{2} \]

Since our choice of \( M \) was arbitrary, we see that statement (2) holds for any TM \( M \). In particular, this means that statement (2) holds for the TM \( R \), which tells us that

\[ \langle R \rangle \notin \mathcal{L}(R) \text{ iff } \langle R \rangle \in \mathcal{L}(R). \tag{3} \]

This is clearly impossible. We have reached a contradiction, so our assumption must have been wrong.
Theorem: \( L_D \notin \text{RE} \).

Proof: By contradiction; assume that \( L_D \in \text{RE} \). Then there must be some recognizer \( R \) such that \( \mathcal{L}(R) = L_D \).

Let \( M \) be an arbitrary TM. Since \( \mathcal{L}(R) = L_D \), we know that

\[
\langle M \rangle \in L_D \iff \langle M \rangle \in \mathcal{L}(R). \quad (1)
\]

From the definition of \( L_D \), we see that \( \langle M \rangle \in L_D \iff \langle M \rangle \notin \mathcal{L}(M) \).

Combining this with statement (1) tells us that

\[
\langle M \rangle \notin \mathcal{L}(M) \iff \langle M \rangle \in \mathcal{L}(R). \quad (2)
\]

Since our choice of \( M \) was arbitrary, we see that statement (2) holds for any TM \( M \). In particular, this means that statement (2) holds for the TM \( R \), which tells us that

\[
\langle R \rangle \notin \mathcal{L}(R) \iff \langle R \rangle \in \mathcal{L}(R). \quad (3)
\]

This is clearly impossible. We have reached a contradiction, so our assumption must have been wrong. Thus \( L_D \notin \text{RE} \).
\[ L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathcal{L}(M) \} \]

**Theorem:** \( L_D \notin \text{RE} \).

**Proof:** By contradiction; assume that \( L_D \in \text{RE} \). Then there must be some recognizer \( R \) such that \( \mathcal{L}(R) = L_D \).

Let \( M \) be an arbitrary TM. Since \( \mathcal{L}(R) = L_D \), we know that

\[ \langle M \rangle \in L_D \iff \langle M \rangle \in \mathcal{L}(R). \quad (1) \]

From the definition of \( L_D \), we see that \( \langle M \rangle \in L_D \iff \langle M \rangle \notin \mathcal{L}(M) \).

Combining this with statement (1) tells us that

\[ \langle M \rangle \notin \mathcal{L}(M) \iff \langle M \rangle \in \mathcal{L}(R). \quad (2) \]

Since our choice of \( M \) was arbitrary, we see that statement (2) holds for any TM \( M \). In particular, this means that statement (2) holds for the TM \( R \), which tells us that

\[ \langle R \rangle \notin \mathcal{L}(R) \iff \langle R \rangle \in \mathcal{L}(R). \quad (3) \]

This is clearly impossible. We have reached a contradiction, so our assumption must have been wrong. Thus \( L_D \notin \text{RE} \). ■
What This Means

• On a deeper philosophical level, the fact that non-RE languages exist supports the following claim:

  *There are statements that are true but not provable.*

• Intuitively, given any non-RE language, there will be some string in the language that cannot be proven to be in the language.

• This result can be formalized as a result called Gödel's incompleteness theorem, one of the most important mathematical results of all time.

• Want to learn more? Take Phil 152 or CS154!
What This Means

• On a more philosophical note, you could interpret the previous result in the following way:

  *There are inherent limits about what mathematics can teach us.*

• There's no automatic way to do math. There are true statements that we can't prove.

• That doesn't mean that mathematics is worthless. It just means that we need to temper our expectations about it.
Up to this point:

“Can we solve this problem?”

(Computability Theory)

Starting today:

“Ok, even if we can, we need to consider whether the time/resources required actually make practical/feasible sense.”

(Complexity Theory)
Up to this point:

“Can we solve this problem?”

(Computability Theory)
Up to this point:
“Can we solve this problem?”
(Computability Theory)

Starting today:
“Ok, even if we can, we need to consider whether the time/resources required actually make practical/feasible sense.”
(Complexity Theory)
The class \textbf{R} represents problems that can be solved by a computer.

The class \textbf{RE} represents problems where “yes” answers can be verified by a computer.
Where We're Going

- The class $\mathbf{P}$ represents problems that can be solved *efficiently* by a computer.
- The class $\mathbf{NP}$ represents problems where “yes” answers can be verified *efficiently* by a computer.