For each of the following languages is the language decidable?

- $A_{\text{DFA}} = \{ \langle B, w \rangle | B \text{ is a DFA that accepts } w \}$
- $A_{\text{NFA}} = \{ \langle B, w \rangle | B \text{ is a NFA that accepts } w \}$
For each of the following languages is the language decidable?

• \( A_{DFA} = \{ \langle B, w \rangle | B \text{ is a DFA that accepts } w \} \)
• \( A_{NFA} = \{ \langle B, w \rangle | B \text{ is a NFA that accepts } w \} \)
Turing machines...

\textbf{TM} = Turing machine = program.
Definition
Language $L \subseteq \Sigma^*$ is undecidable if no program $P$, given $w \in \Sigma^*$ as input, can **always stop** and output whether $w \in L$ or $w \notin L$.

(Usually defined using TM not programs. But equivalent.)
**Reminder: Undecidability**

**Definition**

Language $L \subseteq \Sigma^*$ is undecidable if no program $P$, given $w \in \Sigma^*$ as input, can always stop and output whether $w \in L$ or $w \notin L$.

(Usually defined using TM not programs. But equivalent.)
Reminder: Undecidability

Definition

Language $L \subseteq \Sigma^*$ is undecidable if no program $P$, given $w \in \Sigma^*$ as input, can always stop and output whether $w \in L$ or $w \notin L$.

(Usually defined using TM not programs. But equivalent.)
Reminder: The following language is undecidable

Decide if given a program $M$, and an input $w$, does $M$ accepts $w$. Formally, the corresponding language is

$$A_{TM} = \left\{ \langle M, w \rangle \mid M \text{ is a } TM \text{ and } M \text{ accepts } w \right\}.$$
Decide if given a program $M$, and an input $w$, does $M$ accepts $w$. Formally, the corresponding language is

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

**Definition**

A *decider* for a language $L$, is a program (or a TM) that always stops, and outputs for any input string $w \in \Sigma^*$ whether or not $w \in L$.

A language that has a decider is *decidable*. 

Turing proved the following:

**Theorem**

$A_{TM}$ is undecidable.
Decide if given a program $M$, and an input $w$, does $M$ accepts $w$. Formally, the corresponding language is

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

**Definition**
A *decider* for a language $L$, is a program (or a TM) that always stops, and outputs for any input string $w \in \sum^*$ whether or not $w \in L$.

A language that has a decider is *decidable*.

Turing proved the following:

**Theorem**
$A_{TM}$ is undecidable.
The halting problem
$A_{TM}$ is not TM decidable!

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

**Theorem (The halting theorem.)**

$A_{TM}$ is not Turing decidable.
$A_{TM}$ is not TM decidable!

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

**Theorem (The halting theorem.)**  

$A_{TM}$ is not Turing decidable.

**Proof:** Assume $A_{TM}$ is TM decidable...
$A_{TM}$ is not TM decidable!

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

**Theorem (The halting theorem.)**

$A_{TM}$ is not Turing decidable.

**Proof:** Assume $A_{TM}$ is TM decidable...

**Halt:** TM deciding $A_{TM}$. Halt always halts, and works as follows:

$$\text{Halt}(\langle M, w \rangle) = \begin{cases} 
\text{accept} & M \text{ accepts } w \\
\text{reject} & M \text{ does not accept } w.
\end{cases}$$
Halting theorem proof continued 1

We build the following new function:

<table>
<thead>
<tr>
<th>Flipper(⟨M⟩)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ramifications</td>
</tr>
<tr>
<td>res ← Halt(⟨M,M⟩)</td>
</tr>
<tr>
<td>if res is accept then</td>
</tr>
<tr>
<td>reject</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>accept</td>
</tr>
</tbody>
</table>
We build the following new function:

\[
\text{Flipper} (\langle M \rangle) \\
\text{res }\leftarrow\text{Halt}(\langle M, M \rangle) \\
\text{if res is accept then} \\
\quad \text{reject} \\
\text{else} \\
\quad \text{accept}
\]

\text{Flipper always stops:}

\[
\text{Flipper} (\langle M \rangle) = \begin{cases} 
\text{reject} & \text{M accepts} \langle M \rangle \\
\text{accept} & \text{M does not accept} \langle M \rangle.
\end{cases}
\]
Flipper is a TM (duh!), and as such it has an encoding \( \langle \text{Flipper} \rangle \).

Run \text{Flipper} on itself:

\[
\text{Flipper} \left( \langle M \rangle \right) = \begin{cases} 
\text{reject} & M \text{ accepts } \langle M \rangle \\
\text{accept} & M \text{ does not accept } \langle M \rangle.
\end{cases}
\]

This is absurd. Ridiculous even!

Assumption that \text{Halt} exists is false.

\[ \Rightarrow A \text{TM is not TM decidable.} \]
Flipper \( (\langle M \rangle) = \begin{cases} 
\text{reject} & M \text{ accepts } \langle M \rangle \\
\text{accept} & M \text{ does not accept } \langle M \rangle . 
\end{cases} \)

Flipper is a TM (duh!), and as such it has an encoding \( \langle \text{Flipper} \rangle \).

Run Flipper on itself:

Flipper \( (\langle \text{Flipper} \rangle) = \begin{cases} 
\text{reject} & \text{Flipper accepts } \langle \text{Flipper} \rangle \\
\text{accept} & \text{Flipper does not accept } \langle \text{Flipper} \rangle . 
\end{cases} \)

This is absurd. Ridiculous even!
\[ \text{Flipper} \left( \langle M \rangle \right) = \begin{cases} 
\text{reject} & M \text{ accepts } \langle M \rangle \\
\text{accept} & M \text{ does not accept } \langle M \rangle .
\end{cases} \]

\text{Flipper} is a TM (duh!), and as such it has an encoding \( \langle \text{Flipper} \rangle \). Run \text{Flipper} on itself:

\[ \text{Flipper} \left( \langle \text{Flipper} \rangle \right) = \begin{cases} 
\text{reject} & \text{Flipper} \text{ accepts } \langle \text{Flipper} \rangle \\
\text{accept} & \text{Flipper} \text{ does not accept } \langle \text{Flipper} \rangle .
\end{cases} \]

This is absurd. Ridiculous even!

Assumption that \text{Halt} exists is false. \( \implies A_{TM} \) is not TM decidable.
Reductions
Meta definition: Problem $X$ reduces to problem $B$, if given a solution to $B$, then it implies a solution for $X$. Namely, we can solve $Y$ then we can solve $X$. We will done this by $X \implies Y$. 
Meta definition: Problem $X$ reduces to problem $B$, if given a solution to $B$, then it implies a solution for $X$. Namely, we can solve $Y$ then we can solve $X$. We will done this by $X \implies Y$.

Definition
oracle $ORAC$ for language $L$ is a function that receives as a word $w$, returns $\text{TRUE} \iff w \in L$. 
Meta definition: Problem $X$ reduces to problem $B$, if given a solution to $B$, then it implies a solution for $X$. Namely, we can solve $Y$ then we can solve $X$. We will done this by $X \implies Y$.

Definition
oracle $ORAC$ for language $L$ is a function that receives as a word $w$, returns $TRUE \iff w \in L$.

Lemma
A language $X$ reduces to a language $Y$, if one can construct a $TM$ decider for $X$ using a given oracle $ORAC_Y$ for $Y$.

We will denote this fact by $X \implies Y$. 
Reduction proof technique

- **Y**: Problem/language for which we want to prove undecidable.
Reduction proof technique

• **Y**: Problem/language for which we want to prove undecidable.

• Proof via reduction. Result in a proof by contradiction.
Reduction proof technique

- **Y**: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- **L**: language of **Y**.
Reduction proof technique

• **Y**: Problem/language for which we want to prove undecidable.
• Proof via reduction. Result in a proof by contradiction.
• **L**: language of **Y**.
• Assume **L** is decided by **TM M**.
Reduction proof technique

- **Y**: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- **L**: language of **Y**.
- Assume **L** is decided by TM **M**.
- Create a decider for known undecidable problem **X** using **M**.
Reduction proof technique

- **Y**: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- **L**: language of **Y**.
- Assume **L** is decided by TM **M**.
- Create a decider for known undecidable problem **X** using **M**.
- Result in decider for **X** (i.e., **A_{TM}**).
Reduction proof technique

- \(Y\): Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- \(L\): language of \(Y\).
- Assume \(L\) is decided by \(TM\) \(M\).
- Create a decider for known undecidable problem \(X\) using \(M\).
- Result in decider for \(X\) (i.e., \(A_{TM}\)).
- Contradiction \(X\) is not decidable.
Reduction proof technique

- **Y**: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- **L**: language of **Y**.
- Assume **L** is decided by **TM M**.
- Create a decider for known undecidable problem **X** using **M**.
- Result in decider for **X** (i.e., \( A^{TM} \)).
- Contradiction **X** is not decidable.
- Thus, **L** must be not decidable.
**Lemma**

Let $X$ and $Y$ be two languages, and assume that $X \implies Y$. If $Y$ is decidable then $X$ is decidable.

**Proof.**

Let $T$ be a decider for $Y$ (i.e., a program or a TM). Since $X$ reduces to $Y$, it follows that there is a procedure $T_{X|Y}$ (i.e., decider) for $X$ that uses an oracle for $Y$ as a subroutine. We replace the calls to this oracle in $T_{X|Y}$ by calls to $T$. The resulting program $T_X$ is a decider and its language is $X$. Thus $X$ is decidable (or more formally TM decidable).
Lemma
Let $X$ and $Y$ be two languages, and assume that $X \implies Y$. If $X$ is undecidable then $Y$ is undecidable.
Halting
Language of all pairs $\langle M, w \rangle$ such that $M$ halts on $w$:

$$A_{\text{Halt}} = \left\{ \langle M, w \rangle \mid M \text{ is a } TM \text{ and } M \text{ stops on } w \right\}.$$ 

Similar to language already known to be undecidable:

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

The language $A_{TM}$ reduces to $A_{Halt}$. Namely, given an oracle for $A_{Halt}$ one can build a decider (that uses this oracle) for $A_{TM}$.
On way to proving that Halting is undecidable...

Proof.
Let $\text{ORAC}_{\text{Halt}}$ be the given oracle for $\text{A}_{\text{Halt}}$. We build the following decider for $\text{A}_{\text{TM}}$.

AnotherDecider-$\text{A}_{\text{TM}}(\langle M, w \rangle)$

res ← $\text{ORAC}_{\text{Halt}}(\langle M, w \rangle)$

// if $M$ does not halt on $w$ then reject.
if res = reject then
    halt and reject.

// $M$ halts on $w$ since res = accept.
// Simulating $M$ on $w$ terminates in finite time.
res$_2$ ← Simulate $M$ on $w$.

return res$_2$.

This procedure always return and as such its a decider for $\text{A}_{\text{TM}}$. □
The Halting problem is not decidable

**Theorem**
The language $A_{\text{Halt}}$ is not decidable.

**Proof.**
Assume, for the sake of contradiction, that $A_{\text{Halt}}$ is decidable. As such, there is a $\text{TM}$, denoted by $\text{TM}_{\text{Halt}}$, that is a decider for $A_{\text{Halt}}$. We can use $\text{TM}_{\text{Halt}}$ as an implementation of an oracle for $A_{\text{Halt}}$, which would imply that one can build a decider for $A_{\text{TM}}$. However, $A_{\text{TM}}$ is undecidable. A contradiction. It must be that $A_{\text{Halt}}$ is undecidable. □
The same proof by figure...

... if $A_{Halt}$ is decidable, then $A_{TM}$ is decidable, which is impossible.
Emptiness
The language of empty languages

- $E_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset \}$. 
- $TM_{ETM}$: Assume we are given this decider for $E_{TM}$.
- Need to use $TM_{ETM}$ to build a decider for $A_{TM}$.
- Decider for $A_{TM}$ is given $M$ and $w$ and must decide whether $M$ accepts $w$.
- Restructure question to be about Turing machine having an empty language.
- Somehow make the second input ($w$) disappear.
The language of empty languages

- \( E_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset \} \).
- \( TM_{ETM} \): Assume we are given this decider for \( E_{TM} \).
- Need to use \( TM_{ETM} \) to build a decider for \( A_{TM} \).
- Decider for \( A_{TM} \) is given \( M \) and \( w \) and must decide whether \( M \) accepts \( w \).
- Restructure question to be about Turing machine having an empty language.
- Somehow make the second input (\( w \)) disappear.
- Idea: hard-code \( w \) into \( M \), creating a TM \( M_w \) which runs \( M \) on the fixed string \( w \).
- TM \( M_w \):
  1. Input = \( x \) (which will be ignored)
  2. Simulate \( M \) on \( w \).
  3. If the simulation accepts, accept. If the simulation rejects, reject.
Embedding strings...

- Given program \( \langle M \rangle \) and input \( w \)...
- ...can output a program \( \langle M_w \rangle \).
- The program \( M_w \) simulates \( M \) on \( w \). And accepts/rejects accordingly.
- \textbf{EmbedString}(\( \langle M, w \rangle \)) input two strings \( \langle M \rangle \) and \( w \), and output a string encoding (TM) \( \langle M_w \rangle \).
• Given program $\langle M \rangle$ and input $w$...
• ...can output a program $\langle M_w \rangle$.
• The program $M_w$ simulates $M$ on $w$. And accepts/rejects accordingly.

• EmbedString$(\langle M, w \rangle)$ input two strings $\langle M \rangle$ and $w$, and output a string encoding (TM) $\langle M_w \rangle$.

• What is $L(M_w)$?
• Given program \( \langle M \rangle \) and input \( w \)...
• ...can output a program \( \langle M_w \rangle \).
• The program \( M_w \) simulates \( M \) on \( w \). And accepts/rejects accordingly.
• \textbf{EmbedString}(\( \langle M, w \rangle \)) input two strings \( \langle M \rangle \) and \( w \), and output a string encoding (TM) \( \langle M_w \rangle \).
• What is \( L(M_w) \)?
• Since \( M_w \) ignores input \( x \). language \( M_w \) is either \( \Sigma^* \) or \( \emptyset \). It is \( \Sigma^* \) if \( M \) accepts \( w \), and it is \( \emptyset \) if \( M \) does not accept \( w \).
Theorem
The language $E_{TM}$ is undecidable.

- Assume (for contradiction), that $E_{TM}$ is decidable.
- $TM_{ETM}$ be its decider.
- Build decider $\text{AnotherDecider-}A_{TM}$ for $A_{TM}$:

$$\begin{align*}
\text{AnotherDecider-}A_{TM}(\langle M, w \rangle) \\
\langle M_w \rangle &\leftarrow \text{EmbedString}(\langle M, w \rangle) \\
r &\leftarrow TM_{ETM}(\langle M_w \rangle). \\
\text{if } r = \text{accept} \text{ then} \\
&\quad \text{return reject} \\
// TM_{ETM}(\langle M_w \rangle) \text{ rejected its input} \\
\text{return accept}
\end{align*}$$
Emptiness is undecidable...

Consider the possible behavior of AnotherDecider-$A_{TM}$ on the input $\langle M, w \rangle$.

- If $TM_{ETM}$ accepts $\langle M_w \rangle$, then $L(M_w)$ is empty. This implies that $M$ does not accept $w$. As such, AnotherDecider-$A_{TM}$ rejects its input $\langle M, w \rangle$.
- If $TM_{ETM}$ accepts $\langle M_w \rangle$, then $L(M_w)$ is not empty. This implies that $M$ accepts $w$. So AnotherDecider-$A_{TM}$ accepts $\langle M, w \rangle$. 

...must be assumption that $E_{TM}$ is decidable is false.
Consider the possible behavior of $\text{AnotherDecider-}A_{TM}$ on the input $\langle M, w \rangle$.

- If $TM_{ETM}$ accepts $\langle M_w \rangle$, then $L(M_w)$ is empty. This implies that $M$ does not accept $w$. As such, $\text{AnotherDecider-}A_{TM}$ rejects its input $\langle M, w \rangle$.
- If $TM_{ETM}$ accepts $\langle M_w \rangle$, then $L(M_w)$ is not empty. This implies that $M$ accepts $w$. So $\text{AnotherDecider-}A_{TM}$ accepts $\langle M, w \rangle$.

$\implies \text{AnotherDecider-}A_{TM}$ is decider for $A_{TM}$.

But $A_{TM}$ is undecidable...
Emptiness is undecidable...

Consider the possible behavior of $\text{AnotherDecider-}A_{TM}$ on the input $\langle M, w \rangle$.

- If $TM_{ETM}$ accepts $\langle M_w \rangle$, then $L(M_w)$ is empty. This implies that $M$ does not accept $w$. As such, $\text{AnotherDecider-}A_{TM}$ rejects its input $\langle M, w \rangle$.

- If $TM_{ETM}$ accepts $\langle M_w \rangle$, then $L(M_w)$ is not empty. This implies that $M$ accepts $w$. So $\text{AnotherDecider-}A_{TM}$ accepts $\langle M, w \rangle$.

$\implies \text{AnotherDecider-}A_{TM}$ is decider for $A_{TM}$.

But $A_{TM}$ is undecidable...

...must be assumption that $E_{TM}$ is decidable is false.
AnotherDecider-\(A_{TM}\) never actually runs the code for \(M_w\). It hands the code to a function \(TM_{ETM}\) which analyzes what the code would do if run it. So it does not matter that \(M_w\) might go into an infinite loop.
Equality
Equality is undecidable

\[ EQ_{TM} = \{ \langle M, N \rangle \mid M \text{ and } N \text{ are } TM's \text{ and } L(M) = L(N) \} . \]

**Lemma**

The language \( EQ_{TM} \) is undecidable.
Proof.
Suppose that we had a decider $\text{DeciderEqual}$ for $EQ_{TM}$. Then we can build a decider for $E_{TM}$ as follows:

$TM \; R$:  
1. Input = $\langle M \rangle$
2. Include the (constant) code for a $TM$ $T$ that rejects all its input. We denote the string encoding $T$ by $\langle T \rangle$.
3. Run $\text{DeciderEqual}$ on $\langle M, T \rangle$.
4. If $\text{DeciderEqual}$ accepts, then accept.
5. If $\text{DeciderEqual}$ rejects, then reject.
Regularity
Many undecidable languages

• Almost any property defining a TM language induces a language which is undecidable.
• proofs all have the same basic pattern.
• Regularity language:
  \[ \text{Regular}_{\text{TM}} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular} \} \].
• \textbf{DeciderRegL}: Assume TM decider for \text{Regular}_{\text{TM}}.
• Reduction from halting requires to turn problem about deciding whether a TM \( M \) accepts \( w \) (i.e., is \( w \in A_{\text{TM}} \)) into a problem about whether some TM accepts a regular set of strings.
Proof continued...

• Given $M$ and $w$, consider the following TM $M'_w$:

  **TM $M'_w$:**
  (i) Input = $x$
  (ii) If $x$ has the form $a^n b^n$, halt and accept.
  (iii) Otherwise, simulate $M$ on $w$.
  (iv) If the simulation accepts, then accept.
  (v) If the simulation rejects, then reject.

• **not** executing $M'_w$!

• feed string $\langle M'_w \rangle$ into **DeciderRegL**

• **EmbedRegularString**: program with input $\langle M \rangle$ and $w$, and outputs $\langle M'_w \rangle$, encoding the program $M'_w$.

• If $M$ accepts $w$, then any $x$ accepted by $M'_w$: $L(M'_w) = \Sigma^*$.

• If $M$ does not accept $w$, then $L(M'_w) = \{a^n b^n \mid n \geq 0\}$. 

28
Proof continued...

• $a^n b^n$ is not regular...
• Use \textbf{DeciderRegL} on $M'_w$ to distinguish these two cases.
• Note - cooked $M'_w$ to the decider at hand.
• A decider for $A_{TM}$ as follows.

$$\text{AnotherDecider-}A_{TM}(\langle M, w \rangle)$$

$$\langle M'_w \rangle \leftarrow \text{EmbedRegularString}(\langle M, w \rangle)$$

$$r \leftarrow \text{DeciderRegL}(\langle M'_w \rangle).$$

\[\text{return } r\]

• If \textbf{DeciderRegL} accepts $\implies L(M'_w)$ regular (its $\Sigma^*$)
Proof continued...

- \(a^n b^n\) is not regular...
- Use \textbf{DeciderRegL} on \(M'_w\) to distinguish these two cases.
- Note - cooked \(M'_w\) to the decider at hand.
- A decider for \(A_{TM}\) as follows.

\[
\text{AnotherDecider-} A_{TM}(\langle M, w \rangle)
\]
\[
\langle M'_w \rangle \leftarrow \text{EmbedRegularString}(\langle M, w \rangle)
\]
\[
r \leftarrow \text{DeciderRegL}(\langle M'_w \rangle).
\]
\[
\text{return } r
\]

- If \textbf{DeciderRegL} accepts \( \Rightarrow L(M'_w) \) regular (its \( \Sigma^* \)) \( \Rightarrow M \) accepts \( w \). So \textbf{AnotherDecider-} \( A_{TM} \) should accept \( \langle M, w \rangle \).
• $a^n b^n$ is not regular...
• Use DeciderRegL on $M'_w$ to distinguish these two cases.
• Note - cooked $M'_w$ to the decider at hand.
• A decider for $A_{TM}$ as follows.

```
AnotherDecider-A_{TM}(⟨M, w⟩)
⟨M'_w⟩ ← EmbedRegularString(⟨M, w⟩)
  \( r ← \text{DeciderRegL}(⟨M'_w⟩). \)

return \( r \)
```
• If \( \text{DeciderRegL} \) accepts $\implies L(M'_w)$ regular (its $\Sigma^*$) $\implies M$ accepts $w$. So AnotherDecider-A_{TM} should accept $⟨M, w⟩$.
• If \( \text{DeciderRegL} \) rejects $\implies L(M'_w)$ is not regular $\implies L(M'_w) = a^n b^n$
• $a^n b^n$ is not regular...
• Use **DeciderRegL** on $M'_w$ to distinguish these two cases.
• Note - cooked $M'_w$ to the decider at hand.
• A decider for $A_{TM}$ as follows.

$$\text{AnotherDecider-}A_{TM}(\langle M, w \rangle)$$

$$\langle M'_w \rangle \leftarrow \text{EmbedRegularString}(\langle M, w \rangle)$$

$$r \leftarrow \text{DeciderRegL}(\langle M'_w \rangle).$$

return $r$

• If **DeciderRegL** accepts $\implies L(M'_w)$ regular (its $\Sigma^*$) $\implies M$ accepts $w$. So **AnotherDecider-** $A_{TM}$ should accept $\langle M, w \rangle$.
• If **DeciderRegL** rejects $\implies L(M'_w)$ is not regular $\implies L(M'_w) = a^n b^n$ $\implies M$ does not accept $w$ $\implies$ **AnotherDecider-** $A_{TM}$ should reject $\langle M, w \rangle$. 
The above proofs were somewhat repetitious...

...they imply a more general result.

**Theorem (Rice’s Theorem.)**

Suppose that $L$ is a language of Turing machines; that is, each word in $L$ encodes a TM. Furthermore, assume that the following two properties hold.

(a) Membership in $L$ depends only on the Turing machine’s language, i.e. if $L(M) = L(N)$ then $\langle M \rangle \in L \iff \langle N \rangle \in L$.

(b) The set $L$ is “non-trivial,” i.e. $L \neq \emptyset$ and $L$ does not contain all Turing machines.

Then $L$ is an undecidable.