Undecidability II: More problems via reductions

Lecture 21
Thursday, November 16, 2017
Turing machines...

$\text{TM} = \text{Turing machine} = \text{program}.$
Definition 1

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.)
Definition 1

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.)
Definition 1

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 \( \text{TM} \) 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

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

**Definition 2**

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 3**

$A_{TM}$ is undecidable.
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

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

**Definition 2**

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 3**

$A_{\text{TM}}$ is undecidable.
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

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

**Definition 2**

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 3**

$A_{TM}$ is undecidable.
Part I

Reductions
Meta definition: Problem \textbf{A reduces} to problem \textbf{B}, if given a solution to \textbf{B}, then it implies a solution for \textbf{A}. Namely, we can solve \textbf{B} then we can solve \textbf{A}. We will done this by \textbf{A} \implies \textbf{B}.

Definition 4
oracle \textbf{ORAC} for language \textbf{L} is a function that receives as a word \textbf{w}, returns \textbf{TRUE} \iff \textbf{w} \in \textbf{L}.

Definition 5
\textbf{A language X reduces} to a language \textbf{Y}, if one can construct a \textbf{TM} decider for \textbf{X} using a given oracle \textbf{ORAC}_\textbf{Y} for \textbf{Y}. We will denote this fact by \textbf{X} \implies \textbf{Y}.
**Reduction**

**Meta definition:** Problem \( A \) **reduces** to problem \( B \), if given a solution to \( B \), then it implies a solution for \( A \). Namely, we can solve \( B \) then we can solve \( A \). We will done this by \( A \implies B \).

**Definition 4**

**oracle** \( \text{ORAC} \) for language \( L \) is a function that receives as a word \( w \), returns \( \text{TRUE} \iff w \in L \).

**Definition 5**

A language \( X \) **reduces** to a language \( Y \), if one can construct a \( \text{TM} \) decider for \( X \) using a given oracle \( \text{ORAC}_Y \) for \( Y \). We will denote this fact by \( X \implies Y \).
**Reduction**

**Meta definition:** Problem **A reduces** to problem **B**, if given a solution to **B**, then it implies a solution for **A**. Namely, we can solve **B** then we can solve **A**. We will done this by **A \implies B**.

**Definition 4**

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

**Definition 5**

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

1. **B**: Problem/language for which we want to prove undecidable.
3. **L**: language of **B**.
4. Assume **L** is decided by **TM M**.
5. Create a decider for known undecidable problem **A** using **M**.
6. Result in decider for **A** (i.e., **A_{TM}**).
7. Contradiction **A** is not decidable.
8. Thus, **L** must be not decidable.
Reduction proof technique

1. **B**: Problem/language for which we want to prove undecidable.
3. **L**: Language of **B**.
4. Assume **L** is decided by **TM** **M**.
5. Create a decider for known undecidable problem **A** using **M**.
6. Result in decider for **A** (i.e., **A**<sub>TM</sub>).
7. Contradiction **A** is not decidable.
8. Thus, **L** must be not decidable.
Reduction proof technique

1. **B**: Problem/language for which we want to prove undecidable.


3. **L**: language of **B**.

4. Assume **L** is decided by **TM M**.

5. Create a decider for known undecidable problem **A** using **M**.

6. Result in decider for **A** (i.e., **A_{TM}**).

7. Contradiction **A** is not decidable.

8. Thus, **L** must be not decidable.
Reduction proof technique

1. **B**: Problem/language for which we want to prove undecidable.
3. **L**: language of B.
4. Assume L is decided by TM M.
5. Create a decider for known undecidable problem A using M.
6. Result in decider for A (i.e., A_{TM}).
7. Contradiction A is not decidable.
8. Thus, L must be not decidable.
Reduction proof technique

1. **B**: Problem/language for which we want to prove undecidable.
3. \( L \): language of \( B \).
4. Assume \( L \) is decided by \( TM \ M \).
5. Create a decider for known undecidable problem \( A \) using \( M \).
6. Result in decider for \( A \) (i.e., \( A_{TM} \)).
7. Contradiction \( A \) is not decidable.
8. Thus, \( L \) must be not decidable.
Reduction proof technique

1. **B**: Problem/language for which we want to prove undecidable.
3. **L**: language of **B**.
4. Assume **L** is decided by **TM M**.
5. Create a decider for known undecidable problem **A** using **M**.
6. Result in decider for **A** (i.e., **A_{TM}**).
7. Contradiction **A** is not decidable.
8. Thus, **L** must be not decidable.
Reduction proof technique

1. **B**: Problem/language for which we want to prove undecidable.
3. **L**: language of **B**.
4. Assume **L** is decided by **TM M**.
5. Create a decider for known undecidable problem **A** using **M**.
6. Result in decider for **A** (i.e., **A_{TM}**).
7. Contradiction **A** is not decidable.
8. Thus, **L** must be not decidable.
Reduction proof technique

1. **B**: Problem/language for which we want to prove undecidable.
3. **L**: language of **B**.
4. Assume **L** is decided by **TM** **M**.
5. Create a decider for known undecidable problem **A** using **M**.
6. Result in decider for **A** (i.e., **A_{TM}**).
7. Contradiction **A** is not decidable.
8. Thus, **L** must be not decidable.
Lemma 6

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 7

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

Halting
The halting problem

Language of all pairs $\langle M, w \rangle$ such that $M$ halts on $w$: 

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

Similar to language already known to be undecidable: 

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

Language of all pairs $\langle M, w \rangle$ such that $M$ halts on $w$:

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

Similar to language already known to be undecidable:

$$A_{\text{TM}} = \left\{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \right\}.$$
On way to proving that Halting is undecidable...

**Lemma 8**

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 $A_{TM}$.
Proof of lemma

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

\[
\text{Decider-}A_{\text{TM}}\left(\langle M, w \rangle \right)
\]
\[
\begin{align*}
\text{res} & \leftarrow \text{ORAC}_{\text{Halt}}\left(\langle M, w \rangle \right) \\
\text{if } \text{res} = \text{reject} & \text{ then} \\
& \text{halt and reject.} \\
\text{if } \text{M} \text{ does not halt on } w & \text{ then reject.} \\
\text{else } \text{M} \text{ halts on } w & \text{ since } \text{res} = \text{accept.} \\
& \text{Simulating M on w terminates in finite time.} \\
\text{res}_2 & \leftarrow \text{Simulate M on w.} \\
\text{return } \text{res}_2.
\end{align*}
\]

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

Theorem 9

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 TM, denoted by $TM_{\text{Halt}}$, that is a decider for $A_{\text{Halt}}$. We can use $TM_{\text{Halt}}$ as an implementation of an oracle for $A_{\text{Halt}}$, which would imply by Lemma ?? 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.
... if $A_{\text{Halt}}$ is decidable, then $A_{\text{TM}}$ is decidable, which is impossible.
Part III

Emptiness
The language of empty languages

1. \( E_{\text{TM}} = \left\{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset \right\} \).

2. \( TM_{ETM} \): Assume we are given this decider for \( E_{\text{TM}} \).

3. Need to use \( TM_{ETM} \) to build a decider for \( A_{\text{TM}} \).

4. Decider for \( A_{\text{TM}} \) is given \( M \) and \( w \) and must decide whether \( M \) accepts \( w \).

5. Restructure question to be about Turing machine having an empty language.

6. Somehow make the second input (\( w \)) disappear.

7. Idea: hard-code \( w \) into \( M \), creating a \( TM \) \( M_w \) which runs \( M \) on the fixed string \( w \).

8. \( TM \) \( M_w \):
   1. Input = \( x \) (which will be ignored)
   2. Simulate \( M \) on \( w \).
   3. If the simulation accepts, accept. If the simulation rejects, reject.
The language of empty languages

1. \( E_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset \} \).

2. \( TM_{ETM} \): Assume we are given this decider for \( E_{TM} \).

3. Need to use \( TM_{ETM} \) to build a decider for \( A_{TM} \).

4. Decider for \( A_{TM} \) is given \( M \) and \( w \) and must decide whether \( M \) accepts \( w \).

5. Restructure question to be about Turing machine having an empty language.

6. Somehow make the second input \((w)\) disappear.

7. Idea: hard-code \( w \) into \( M \), creating a \( TM \) \( M_w \) which runs \( M \) on the fixed string \( w \).

8. \( TM \) \( M_w \):
   1. Input = \( x \) (which will be ignored)
   2. Simulate \( M \) on \( w \).
   3. If the simulation accepts, accept. If the simulation rejects, reject.
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 ($\text{TM}$) $\langle M_w \rangle$.

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 $w$. 
Embedding strings...

1. Given program $\langle M \rangle$ and input $w$...
2. ...can output a program $\langle M_w \rangle$.
3. The program $M_w$ simulates $M$ on $w$. And accepts/rejects accordingly.
4. **EmbedString($\langle M, w \rangle$)** input two strings $\langle M \rangle$ and $w$, and output a string encoding ($\text{TM}$) $\langle M_w \rangle$.
5. What is $L(M_w)$?
6. 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 $w$. 
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 \( L(M_w) \)?

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 10

The language $E_{TM}$ is undecidable.

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

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

$\langle M_w \rangle \leftarrow EmbedString(\langle M, w \rangle)$

$r \leftarrow TM_{ETM}(\langle M_w \rangle)$.

if $r = \text{accept}$ then

    return reject

// $TM_{ETM}(\langle M_w \rangle)$ rejected its input

return accept
Emptiness is undecidable...

Proof continued

Consider the possible behavior of $\text{AnotherDecider-ATM}$ 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, $\text{AnotherDecider-ATM}$ 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 $\text{AnotherDecider-ATM}$ accepts $\langle M, w \rangle$.

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

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

...must be assumption that $E_{TM}$ is decidable is false.
Emptiness is undecidable...

Proof continued

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

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

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

\(\implies \) \textbf{AnotherDecider-}\textsubscript{\(A_{TM}\)} is decider for \(A_{TM}\).

But \(A_{TM}\) is undecidable...

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

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

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

...must be assumption that $E_{\text{TM}}$ 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.
Part IV

Equality
Equality is undecidable

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

**Lemma 11**

The language \( EQ_{TM} \) is undecidable.
Proof.

Suppose that we had a decider \textbf{DeciderEqual} for $EQ_{TM}$. Then we can build a decider for $E_{TM}$ as follows:

\textbf{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 \textbf{DeciderEqual} on $\langle M, T \rangle$.
4. If \textbf{DeciderEqual} accepts, then accept.
5. If \textbf{DeciderEqual} rejects, then reject.
Part V

Regularity
Many undecidable languages

1. Almost any property defining a TM language induces a language which is undecidable.
2. Proofs all have the same basic pattern.
3. Regularity language:
   \[ \text{Regular}_\text{TM} = \left\{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular} \right\} . \]
4. DeciderRegL: Assume TM decider for Regular_\text{TM}.
5. Reduction from halting requires to turn problem about deciding whether a TM M accepts w (i.e., is w ∈ A_\text{TM}) into a problem about whether some TM accepts a regular set of strings.
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.

2 **not** executing $M'_w$!

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

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

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

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

2. Use $\text{DeciderRegL}$ on $M'_w$ to distinguish these two cases.

3. Note - cooked $M'_w$ to the decider at hand.

4. A decider for $A_{TM}$ as follows.

   YetAnotherDecider-$A_{TM}(⟨M, w⟩)$
   $⟨M'_w⟩ \leftarrow \text{EmbedRegularString}(⟨M, w⟩)$
   $r \leftarrow \text{DeciderRegL}(⟨M'_w⟩)$.
   return $r$

5. If $\text{DeciderRegL}$ accepts $\implies L(M'_w)$ regular (its $Σ^*$) $\implies M$ accepts $w$. So YetAnotherDecider-$A_{TM}$ should accept $⟨M, w⟩$.

6. If $\text{DeciderRegL}$ rejects $\implies L(M'_w)$ is not regular $\implies L(M'_w) = a^n b^n \implies M$ does not accept $w$ $\implies$ YetAnotherDecider-$A_{TM}$ should reject $⟨M, w⟩$. 
Proof continued...

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

\[
\begin{align*}
\text{YetAnotherDecider-} \text{A}_{\text{TM}} (\langle M, w \rangle) & \\
\langle M'_w \rangle & \leftarrow \text{EmbedRegularString} (\langle M, w \rangle) \\
r & \leftarrow \text{DeciderRegL} (\langle M'_w \rangle).
\end{align*}
\]

return \(r\)

5. If \(\text{DeciderRegL}\) accepts \(\implies L(M'_w)\) regular \((\text{its } \Sigma^*) \implies M\) accepts \(w\). So \text{YetAnotherDecider-} \text{A}_{\text{TM}}\) should accept \(\langle M, w \rangle\).

6. If \(\text{DeciderRegL}\) rejects \(\implies L(M'_w)\) is not regular \(\implies L(M'_w) = a^n b^n \implies M\) does not accept \(w\) \(\implies \) \text{YetAnotherDecider-} \text{A}_{\text{TM}}\) should reject \(\langle M, w \rangle\).
Proof continued...

1. $a^n b^n$ is not regular...

2. Use $\text{DeciderRegL}$ on $M'_w$ to distinguish these two cases.

3. Note - cooked $M'_w$ to the decider at hand.

4. A decider for $A_{TM}$ as follows.

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

5. If $\text{DeciderRegL}$ accepts $\implies L(M'_w)$ regular (its $\Sigma^*$) $\implies M$ accepts $w$. So $\text{YetAnotherDecider-} A_{TM}$ should accept $\langle M, w \rangle$.

6. If $\text{DeciderRegL}$ rejects $\implies L(M'_w)$ is not regular $\implies L(M'_w) = a^n b^n \implies M$ does not accept $w$ $\implies$ YetAnotherDecider- $A_{TM}$ should reject $\langle M, w \rangle$. 

Sariel Har-Peled (UIUC)
Proof continued...

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

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

5. If \textbf{DeciderRegL} accepts \( \Rightarrow L(M'_w) \) regular (its \( \Sigma^* \)) \( \Rightarrow M \) accepts \( w \). So \textbf{YetAnotherDecider-} A_{TM} should accept \( \langle M, w \rangle \).
6. If \textbf{DeciderRegL} rejects \( \Rightarrow L(M'_w) \) is not regular \( \Rightarrow L(M'_w) = a^n b^n \Rightarrow M \) does not accept \( w \) \( \Rightarrow \textbf{YetAnotherDecider-} A_{TM} \) should reject \( \langle M, w \rangle \).
Rice theorem

The above proofs were somewhat repetitious...
...they imply a more general result.

Theorem 12 (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 undecidable.