Undecidability II: More problems via reductions

Lecture 21
Thursday, April 4, 2019
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 $w \not\in L$.

(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.)
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.
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**.
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.
The following language is undecidable

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

Assume there is a program \( \text{Decide-}A_{\text{TM}}(\langle M \rangle, \omega) \) such that:

\[ \langle M, w \rangle \in A_{\text{TM}} \text{ or } \langle M, w \rangle \notin A_{\text{TM}} \]

if \( M \) accepts \( \omega \).

\( M_{\text{bad}} \):

Input: \( \langle M \rangle \)

- If \( \text{Decide-}A_{\text{TM}}(\langle M \rangle, \langle M \rangle) \) accepts
  
  rejects

  else

  accepts

Contradiction!

\[ \text{Decide-}A_{\text{TM}}(\langle M_{\text{bad}} \rangle, \langle M_{\text{bad}} \rangle) \text{ accept } \Leftrightarrow M_{\text{bad}} \text{ reject } \langle M_{\text{bad}} \rangle \]

\[ \text{Decide-}A_{\text{TM}}(\langle M_{\text{bad}} \rangle, \langle M_{\text{bad}} \rangle) \text{ rejects } \Leftrightarrow M_{\text{bad}} \text{ accepts } \langle M_{\text{bad}} \rangle \]
Part I

Reductions
**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 denote this by $A \implies B$. 
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 denote this by $A \implies B$.

Definition 4

oracle ORAC for language $L$ is a function that receives as a word $w$, returns TRUE $\iff w \in L$. 
**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 denote this by \( A \implies B \).

**Definition 4**

oracle **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 proof technique

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

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

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

Assume \( L \) is decided by \( \text{TM} M \).

Create a decider for known undecidable problem \( A \) using \( M \).

Result in decider for \( A \) (i.e., \( A_{\text{TM}} \)).

Contradiction \( A \) is not decidable.

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**.
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**.
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**\textsuperscript{TM}).
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.
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 implies decidability

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 $\text{TM}$). 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 $\text{TM}$ decidable).
The contrapositive...

**Lemma 7**

Let $X$ and $Y$ be two languages, and assume that $X \iff Y$. 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 $w$:

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

Proof of lemma

Proof.

Let $\text{ORAC}_{\text{Halt}}$ be the given oracle for $A_{\text{Halt}}$. We build the following decider for $A_{\text{TM}}$.
One way to proving that Halting is undecidable...

Proof of lemma

Proof.

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

\[
\text{Decider-} A_{\text{TM}}(\langle M, w \rangle) = \begin{cases} 
\text{reject} & \text{if } \text{res} = \text{reject} \\
\text{accept} & \text{if } \text{res} = \text{accept} 
\end{cases}
\]

Simulating \( M \) on \( w \) terminates in finite time.

\[
\text{res} \leftarrow \text{Simulate} M(\langle M, w \rangle).
\]

return \( \text{res} \).
One way to proving that Halting is undecidable...

Proof of lemma

Proof.

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

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

\[
\begin{align*}
\text{res} & \leftarrow \text{ORAC}_{\text{Halt}}(\langle M, w \rangle) \\
// & \text{ if } M \text{ does not halt on } w \text{ then reject.} \\
\text{if } \text{res} = \text{ reject then} & \\
\text{halt and reject.}
\end{align*}
\]
One way to proving that Halting is undecidable...

Proof of lemma

Proof.

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

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

\begin{align*}
res & \leftarrow \text{ORAC}_{\text{Halt}}(\langle M, w \rangle) \\
// & \text{ if } M \text{ does not halt on } w \text{ then reject.} \\
\text{if } res = \text{ reject} & \text{ then} \\
& \text{ halt and reject.} \\
// & \text{ M halts on } w \text{ since } res = \text{ accept.} \\
// & \text{ Simulating } M \text{ on } w \text{ terminates in finite time.} \\
res_2 & \leftarrow \text{Simulate } M \text{ on } w. \\
\text{return } res_2.
\end{align*}
```

This procedure always return and as such its a decider for $\text{A}_{\text{TM}}$. 

\[\square\]
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 8 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 $A_{\text{Halt}}$ is decidable, then $A_{\text{TM}}$ is decidable, which is impossible.
Part III

Emptiness
The language of empty languages

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

\[ M(x) \]

- Ignore x
- Reject

- \[ L(M) = \emptyset \]
The language of empty languages

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

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

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 \( w \).
The language of empty languages

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

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

3. Need to use \( T M_{ETM} \) 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. Idea: hard-code \( w \) into \( M \), creating a TM \( M_w \) which runs \( M \) on the fixed string \( w \).

6. 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 $⟨M⟩$ and input $w$...

...can output a program $⟨M_w⟩$.

The program $M_w$ simulates $M$ on $w$. And accepts/rejects accordingly.

EmbedString($⟨M, w⟩$) input two strings $⟨M⟩$ and $w$, and output a string encoding (TM) $⟨M_w⟩$.

Since $M_w$ ignores input $x$.. language $M_w$ is either $Σ^∗$ or $∅$. It is $Σ^∗$ if $M$ accepts $w$, and it is $∅$ 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 according to

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.

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 $w$. 
Emptiness is undecidable

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
   
   if $M_w$ rejects all input
   
   return reject
   
   // $TM_{ETM}(\langle M_w \rangle)$ rejected its input
   
   return accept
Consider the possible behavior of $\text{AnotherDecider-} A_{\text{TM}}$ on the input $\langle M, w \rangle$.

- If $T_{M_{\text{ETM}}}$ 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 $T_{M_{\text{ETM}}}$ 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 $\langle M, w \rangle$. 

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 $\langle M, w \rangle$.

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

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

Proof continued

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 $\langle M, w \rangle$.

$\implies \text{AnotherDecider-} A_{\text{TM}}$ 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

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

Lemma 11

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

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

\[ TM \quad R: \]

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 DeciderEqual on \( \langle M, T \rangle \).
4. If DeciderEqual accepts, then accept.
5. If 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.
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:
   \[ \text{Regular}_{TM} = \left\{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular} \right\}. \]
4. **DeciderRegL**: Assume TM decider for Regular_{TM}.
5. Reduction from halting requires to turn problem about deciding whether a TM \( M \) accepts \( w \) (i.e., is \( w \in A_{TM} \)) into a problem about whether some TM accepts a regular set of strings.
Given $M$ and $w$, consider the following TM $M'_w$:

$M'_w$:

(i) Input $= x$

(ii) If $x$ has the form $a^n b^n$, halt and accept.
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.

**Assume there is a decider that can tell me if $L(M'_w)$ is reg.**
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
Given $M$ and $w$, consider the following TM $M'_w$:

**TM $M'_w$:**
1. Input = $x$
2. If $x$ has the form $a^n b^n$, halt and accept.
3. Otherwise, simulate $M$ on $w$.
4. If the simulation accepts, then accept.
5. If the simulation rejects, then reject.

**Not executing $M'_w$!**

- **Feed string $⟨M'_w⟩$ into DeciderRegL**
- **EmbedRegularString**: program with input $⟨M⟩$ and $w$, and outputs $⟨M'_w⟩$, encoding the program $M'_w$. 

### Proof continued...

If $M$ accepts $w$, then $L(M'_w) = \Sigma^*$.

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

Chan, Har-Peled, Hassanieh (UIUC) CS374 28 Spring 2019 28 / 32
Given $M$ and $w$, consider the following TM $M'_w$:

$M'_w$:

(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 $L(M'_w) = \{a^n b^n \mid n \geq 0\}$. 

Proof continued...

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

2. Use **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.

   ```
   \textbf{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{accept}
   
   \text{return } r .
   ```

5. If **DeciderRegL** accepts $\implies L(M'_w)$ regular (its $\Sigma^*$)
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.

   ```
   \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 \( \text{DeciderRegL} \) accepts \( \iff L(M'_w) \) regular (its \( \Sigma^* \)) \( \iff M \) accepts \( w \). So \( \text{YetAnotherDecider-A}_{TM} \) should accept \( \langle M, w \rangle \).
\(a^n b^n\) is not regular…

1. Use **DeciderRegL** on \(M'_w\) to distinguish these two cases.

2. Note - cooked \(M'_w\) to the decider at hand.

3. 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)}.
   
   \text{return } r
   \]

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

5. If **DeciderRegL** rejects \(\iff L(M'_w) \text{ is not regular} \iff L(M'_w) = a^n b^n\)
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 \(A_{\mathbb{TM}}\) as follows.

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

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

$A = \{ \langle M \rangle \mid L(M) \text{ has a property } \}$
Rice theorem

\[ A = \{ \langle M \rangle \mid M \text{ is TM} \Rightarrow L(M) \text{ has property } P \} \]

Try to show \( \exists \langle M \rangle \in A \)

\[ E_{TM} = \{ \langle M \rangle \mid M \text{ is TM} \Rightarrow L(M) = \emptyset \} \]

\[ L_{TM} = \{ \langle M \rangle \mid M \text{ is TM} \Rightarrow L(M) = \{0, 374\} \} \]

\[ M: \begin{cases} \text{ignore input} & \text{reject} \\ \text{accept} & \text{if } x = 374 \end{cases} \]
Rice theorem

\[
A = \begin{cases} 
\{ <M> \mid M \text{ runs in at most 374 steps} \} \\
\{ <M> \mid M \text{ runs in at least 374 steps} \} \\
\{ <M> \mid M \text{ has 374 states} \} \\
\{ <M> \mid M \text{ moves head left} \}
\end{cases}
\]