Undecidability II: More problems via reductions

Lecture 21
Thursday, April 4, 2019

LaTeXed: April 11, 2019 14:43

Turing machines...

TM = Turing machine = program.

Undecidability

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.

Undecidability

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.

Undecidability

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

$$\mathrm{A}_{\mathrm{TM}} = \left\{ \langle \underline{M, w} \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } \underline{M} \text{ accepts } \underline{w} \right\}.$$

The following language is undecidable

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

$$\mathrm{A}_{\mathrm{TM}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ accepts } w \right\}.$$

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

$$\mathrm{A}_{\mathrm{TM}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ accepts } w \right\}.$$

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

$$\mathbf{A}_{\mathrm{TM}} = \left\{ \langle M, w\rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ accepts } w \right\}.$$

Assume there is a program Decide-$A_{TM}(\langle \underline{M} \rangle, w)$

$\langle M, w\rangle \in A_{TM}$ or $\langle M, w\rangle \notin A_{TM}$

if $\underline{M}$ accepts $\underline{w}$

```
M_bad:
  Input: <M>
  If Decide-A_TM(<M>, <M>) accepts
      rejects
  else
      accepts
```

Contradiction!

Decide-$A_{TM}(\langle M_{bad}\rangle, \langle M_{bad}\rangle)$ accept $\iff$ M_{bad} reject $\langle M_{bad}\rangle$

Decide-$A_{TM}(\langle M_{bad}\rangle, \langle M_{bad}\rangle)$ rejects $\iff$ M_{bad} accepts $\langle M_{bad}\rangle$

Part I

Reductions

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

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 denote this by $\mathbf{A} \implies \mathbf{B}$.

Definition 4

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

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 denote this by $\mathbf{A} \implies \mathbf{B}$.

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* $\mathbf{ORAC}_Y$ *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.
2. Proof via reduction. Result in a proof by contradiction.

Reduction proof technique

1. **B**: Problem/language for which we want to prove undecidable.
2. Proof via reduction. Result in a proof by contradiction.
3. L: language of **B**.

Reduction proof technique

1. **B**: Problem/language for which we want to prove undecidable.
2. Proof via reduction. Result in a proof by contradiction.
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.
2. Proof via reduction. Result in a proof by contradiction.
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.
2. Proof via reduction. Result in a proof by contradiction.
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}).

Reduction proof technique

1. **B**: Problem/language for which we want to prove undecidable.
2. Proof via reduction. Result in a proof by contradiction.
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.
2. Proof via reduction. Result in a proof by contradiction.
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 $\mathtt{T}$ be a decider for Y (i.e., a program or a TM). Since X reduces to Y, it follows that there is a procedure $\mathtt{T}_{X|Y}$ (i.e., decider) for X that uses an oracle for Y as a subroutine. We replace the calls to this oracle in $\mathtt{T}_{X|Y}$ by calls to $\mathtt{T}$. The resulting program $\mathtt{T}_{X}$ is a decider and its language is X. Thus X is decidable (or more formally TM decidable). □

The contrapositive...

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:

$$A_{\mathrm{Halt}} = \left\{ \langle \underline{M}, \underline{w} \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ stops on } w \right\}.$$

The halting problem

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

$$A_{\mathbf{Halt}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ stops on } \underline{w} \right\}.$$

Similar to language already known to be undecidable:

$$\mathbf{A}_{\mathrm{TM}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ accepts } \underline{w} \right\}.$$

On way to proving that Halting is undecidable...

Lemma 8

The language A_{TM} *reduces to* $\boldsymbol{A}_{\mathbf{Halt}}$*. Namely, given an oracle for* $\boldsymbol{A}_{\mathbf{Halt}}$ *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 $\mathbf{ORAC}_{\boldsymbol{Halt}}$ be the given oracle for $\boldsymbol{A}_{\mathbf{Halt}}$. We build the following decider for A_{TM}.

One way to proving that Halting is undecidable...

Proof of lemma

Proof.

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

```
Decider-$\mathrm{A}_{\mathsf{TM}}\big(\langle M, w\rangle\big)$
        $res \leftarrow \mathbf{ORAC}_{\mathit{Halt}}\big(\langle \underline{M}, \underline{w}\rangle\big)$
```

One way to proving that Halting is undecidable...

Proof of lemma

Proof.

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

```
Decider-$\mathrm{A}_{\mathsf{TM}}\big(\langle M, w\rangle\big)$
        $res \leftarrow \mathsf{ORAC}_{\mathit{Halt}}\big(\langle M, w\rangle\big)$
        // if $M$ does not halt on $w$ then reject.
        if $res =$ reject then
                halt and reject.
```

One way to proving that Halting is undecidable...

Proof of lemma

Proof.

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

```
Decider-$\mathrm{A}_{\mathrm{TM}}\big(\langle M, w\rangle\big)$
        $res \leftarrow$ ORAC$_{Halt}\big(\langle M, w\rangle\big)$
        // 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.
        $res_2 \leftarrow$Simulate $M$ on $w$.
        return $res_2$.
```

This procedure always return and as such its a decider for A_{TM}. ☐

The Halting problem is not decidable

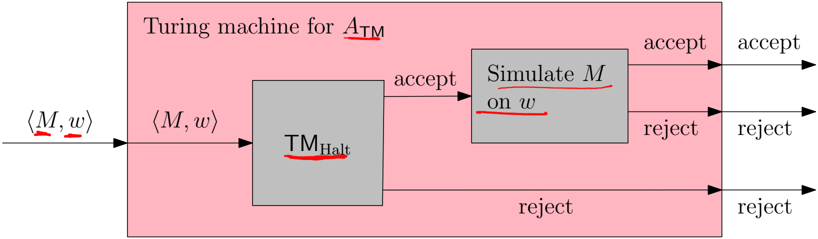
Theorem 9

The language $\boldsymbol{A}_{\mathbf{Halt}}$ is not decidable.

Proof.

Assume, for the sake of contradiction, that $\boldsymbol{A}_{\mathbf{Halt}}$ is decidable. As such, there is a TM, denoted by $\mathrm{TM}_{\mathbf{Halt}}$, that is a decider for $\boldsymbol{A}_{\mathbf{Halt}}$. We can use $\mathrm{TM}_{\mathbf{Halt}}$ as an implementation of an oracle for $\boldsymbol{A}_{\mathbf{Halt}}$, which would imply by Lemma 8 that one can build a decider for A_{TM}. However, A_{TM} is undecidable. A contradiction. It must be that $\boldsymbol{A}_{\mathbf{Halt}}$ is undecidable. □

The same proof by figure...



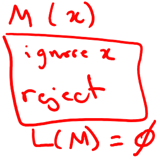
... if $\boldsymbol{A}_{\mathbf{Halt}}$ is decidable, then $\mathbf{A}_{\mathrm{TM}}$ is decidable, which is impossible.

Part III

Emptiness

The language of empty languages

1. $E_{\mathrm{TM}} = \left\{ \langle M \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } L(M) = \emptyset \right\}$.



The language of empty languages

1. $E_{\mathrm{TM}} = \left\{ \langle M \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } L(M) = \emptyset \right\}.$
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. $E_{\mathrm{TM}} = \left\{ \langle M \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } L(M) = \emptyset \right\}.$
2. TM_{ETM}: Assume we are given this decider for E_{TM}.
3. Need to use TM_{ETM} to build a decider for $\mathrm{A_{TM}}$.
4. Decider for $\mathrm{A_{TM}}$ 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.

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**$(\langle \underline{M}, \underline{w} \rangle)$ input two strings $\langle M \rangle$ and w, and output a string encoding (TM) $\langle \underline{M_w} \rangle$.

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 (TM) $\langle M_w \rangle$.
5. What is $\underline{L(M_w)}$?

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 (TM) $\langle M_w \rangle$.
5. What is $L(M_w)$?
6. Since M_w ignores input $\underline{x}$.. language M_w is either Σ^* or $\emptyset$. It is $\underline{\Sigma^*}$ if $\underline{M}$ accepts $\underline{w}$, and it is $\underline{\emptyset}$ if $\underline{M}$ does not accept $\underline{w}$.

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 $r$ = accept then
                return reject
        // $TM_{ETM}(\langle M_w \rangle)$ rejected its input
        return accept
```

if M_w rejects all input

if $L(M_w) = \phi$

Emptiness is undecidable...

Proof continued

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 $\langle M, w \rangle$.

Emptiness is undecidable...

Proof continued

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 $\langle M, w \rangle$.

$\implies$ **AnotherDecider-A_{TM}** is decider for $\mathbf{A}_{\mathrm{TM}}$.

But $\mathbf{A}_{\mathrm{TM}}$ is undecidable...

Emptiness is undecidable...

Proof continued

Consider the possible behavior of **AnotherDecider-$\mathrm{A}_{\mathbf{TM}}$** on the input $\langle M, w\rangle$.

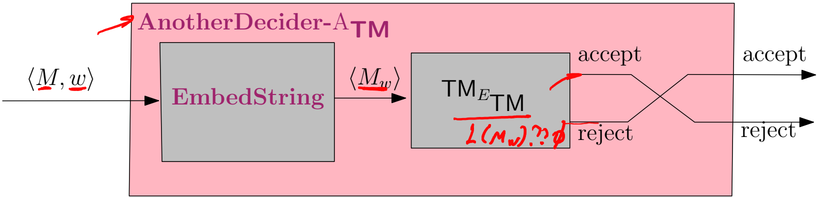
- If TM_{ETM} accepts $\langle M_w\rangle$, then $L(M_w)$ is empty. This implies that M does not accept w. As such, **AnotherDecider-$\mathrm{A}_{\mathbf{TM}}$** rejects its input $\langle M, w\rangle$.
- If TM_{ETM} accepts $\langle M_w\rangle$, then $L(M_w)$ is not empty. This implies that M accepts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

$\Longrightarrow$ **AnotherDecider-$\mathrm{A}_{\mathbf{TM}}$** is decider for $\mathbf{A}_{\mathrm{TM}}$.

But $\mathbf{A}_{\mathrm{TM}}$ is undecidable...

...must be assumption that E_{TM} is decidable is false. ■

Emptiness is undecidable via diagram



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

$$\nearrow EQ_{\mathrm{TM}} = \left\{ \underbrace{\langle \underline{M}, \underline{N} \rangle}_{} \;\middle|\; M \text{ and } N \text{ are } \mathrm{TM}\text{'s and } \underline{L(M)} = \underline{L(N)} \right\}.$$

Lemma 11

The language EQ_{TM} *is undecidable.*

Proof

Proof.

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

TM R:

1. Input $= \langle \underline{M} \rangle$
2. Include the (constant) code for a TM $\textcircled{T}$ 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.

$L(T) = \emptyset$

$L(M) =? L(T)$

$L(M) = \emptyset$

$L(M) \neq \emptyset$

☐

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:

   → $\mathbf{Regular}_{\mathrm{TM}} = \left\{ \langle M \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } \underline{L(M)} \text{ is regular} \right\}.$
4. **DeciderRegL**: Assume TM decider for $\mathbf{Regular}_{\mathrm{TM}}$.
5. Reduction from halting requires to turn problem about deciding whether a TM M accepts w (i.e., is $\underline{w \in \mathrm{A}_{\mathrm{TM}}}$) into a problem about whether some TM accepts a regular set of strings.

Proof continued...

1. Given M and w, consider the following TM $\underbrace{M'_w}$:

TM M'_w:

(i) Input = $\underline{x}$

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

Proof continued...

1. 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 $\underline{M}$ on $\underline{w}$.

   (iv) If the simulation accepts, then accept. ✓

   (v) If the simulation rejects, then reject. ✗

Assm There is a decider that can tell me if $L(M'_w)$ is reg.

Proof continued...

1. 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**

Proof continued...

1. 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.

if M accepts w

M'_w accepts $\Sigma^* \leftarrow$ regu

if M rejects w

M'_w accept $a^n b^n \leftarrow$ regu

Proof continued...

1. 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 $\underline{M}$ on $\underline{w}$.

   (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 $\mathrm{A_{TM}}$ as follows.

```
YetAnotherDecider-A_TM(⟨M, w⟩)
        ⟨M'_w⟩ ← EmbedRegularString (⟨M, w⟩)
        r ← DeciderRegL(⟨M'_w⟩).accept
        return r.
```

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

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.

```
YetAnotherDecider-$\mathrm{A}_{\mathrm{TM}}(\langle M, w \rangle)$
        $\langle M'_w \rangle \leftarrow$ EmbedRegularString $(\langle M, w \rangle)$
        $r \leftarrow$ DeciderRegL$(\langle M'_w \rangle)$.
        return $r$
```

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

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 $\mathbf{A}_{\mathrm{TM}}$ as follows.

```
YetAnotherDecider-A_TM(⟨M, w⟩)
        $\langle M'_w \rangle \leftarrow$ EmbedRegularString(⟨M, w⟩)
        $r \leftarrow$ DeciderRegL(⟨M'_w⟩).
        return r
```

5. If **DeciderRegL** accepts $\implies L(M'_w)$ regular (its Σ^*) $\implies M$ accepts w. So **YetAnotherDecider-$\mathbf{A}_{\mathbf{TM}}$** should accept $\langle M, w \rangle$.
6. If **DeciderRegL** rejects $\implies L(M'_w)$ is not regular $\implies L(M'_w) = a^n b^n$

Proof continued...

1. $\underline{a^n b^n}$ 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.

```
YetAnotherDecider-$\mathrm{A}_{\mathrm{TM}}$($\langle M, w \rangle$)
    $\langle M'_w \rangle \leftarrow$ EmbedRegularString ($\langle M, w \rangle$)
    $r \leftarrow$ DeciderRegL($\langle M'_w \rangle$).
    return $r$
```

5. If **DeciderRegL** accepts $\implies L(M'_w)$ regular (its Σ^*) $\implies M$ accepts w. So **YetAnotherDecider-A_{TM}** should accept $\langle M, w \rangle$.
6. If **DeciderRegL** rejects $\implies L(M'_w)$ is not regular $\implies L(M'_w) = a^n b^n \implies \underline{M \text{ does not accept } w} \implies$ **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* $\mathbf{L(M) = L(N)}$ *then* $\langle M \rangle \in \mathbf{L} \Leftrightarrow \langle N \rangle \in \mathbf{L}$.

(b) *The set* **L** *is "non-trivial," i.e.* $\mathbf{L} \neq \emptyset$ *and* **L** *does not contain all Turing machines.*

Then **L** *is a undecidable.*

→ $\underline{A} = \{ \langle M \rangle \mid L(M) \text{ has a property } P \}$ ✓

Rice theorem

→ $A = \{ \langle M \rangle \mid M \text{ is TM } \& \ \underline{L(M) \text{ has property } P} \}$

eng to show 1 eg. $\langle M \rangle \notin A$

1 eg $\langle M \rangle \in A$ ✓

$E_{TM} = \{ \langle M \rangle \mid M \text{ is TM } \& \ L(M) = \emptyset \}$.

```
M : x
  ignore input
  reject
```

```
M : x
  ignore input
  accept
```

$L_{TM} = \{ \langle M \rangle \mid M \text{ is TM } \& \ L(M) = \{0^{374}\} \}$

```
M: x
  if $x = 0^{374}$
    accept
  else reject
```

Rice theorem

decidable

A = {<M> | M runs in at most 374 steps}

A = {<M> | M runs in at lest 374 steps}

A = {<M> | M has 374 states}

A = {<M> | M moves head to left.}