Undecidability II: More problems via reductions

Lecture 21

Thursday, April 4, 2019

LaTeXed: December 27, 2018 08:26

Turing machines...

TM = Turing machine = program.

Reminder: 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.

Reminder: Undecidability

Definition 1

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

(Usually defined using TM not programs. But equivalent.

Reminder: 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.

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

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

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

$\mathrm{A_{TM}}$ *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

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

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

$\mathrm{A_{TM}}$ *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

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

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

$\mathrm{A_{TM}}$ *is undecidable.*

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 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* $\mathbf{ORAC}_Y$ *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* $\mathbf{ORAC}_Y$ *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 $\mathbf{A} \implies \mathbf{B}$.

Definition 4

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

Definition 5

A language $\boldsymbol{X}$ ***reduces*** *to a language* $\boldsymbol{Y}$*, if one can construct a* TM *decider for* $\boldsymbol{X}$ *using a given oracle* $\mathbf{ORAC_Y}$ *for* $\boldsymbol{Y}$*.*
We will denote this fact by $\boldsymbol{X} \implies \boldsymbol{Y}$*.*

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 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 proof technique

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

The countrapositive...

Lemma 7

Let $\mathbf{X}$ and $\mathbf{Y}$ be two languages, and assume that $\mathbf{X} \implies \mathbf{Y}$. If $\mathbf{X}$ is undecidable then $\mathbf{Y}$ is undecidable.

Part II

Halting

The halting problem

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

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

Similar to language already known to be undecidable:

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

The halting problem

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

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

Similar to language already known to be undecidable:

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

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}.

Proof.

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

```
Decider-A_TM(⟨M, w⟩)
        res ← ORAC_Halt(⟨M, w⟩)
        // 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 A_{TM}. □

The Halting problem is not decidable

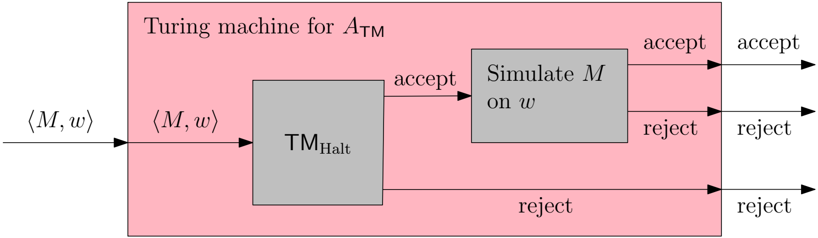
Theorem 9

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. $\square$

The same proof by figure...



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

Part III

Emptiness

The language of empty languages

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

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.
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.

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 x.. language M_w is either Σ^* or $\emptyset$. It is Σ^* 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 (TM) $\langle M_w \rangle$.
5. What is $L(M_w)$?
6. Since M_w ignores input x.. language M_w is either Σ^* or $\emptyset$. It is Σ^* 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 (TM) $\langle M_w \rangle$.
5. What is $L(M_w)$?
6. Since M_w ignores input x.. language M_w is either Σ^* or $\emptyset$. It is Σ^* 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 $r =$ 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 **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 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 **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...

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

Emptiness is undecidable...

Proof continued

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

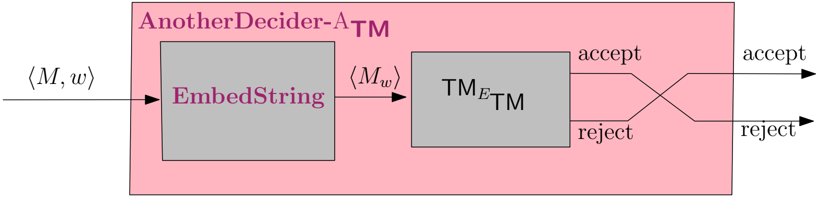
- If $\boldsymbol{TM_{ETM}}$ accepts $\langle \boldsymbol{M_w} \rangle$, then $\boldsymbol{L(M_w)}$ is empty. This implies that $\boldsymbol{M}$ does not accept $\boldsymbol{w}$. As such, **AnotherDecider-A_{TM}** rejects its input $\langle \boldsymbol{M}, \boldsymbol{w} \rangle$.
- If $\boldsymbol{TM_{ETM}}$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

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

But A_{TM} is undecidable...

...must be assumption that $\boldsymbol{E}_{\mathrm{TM}}$ 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

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

Lemma 11

The language $\mathbf{EQ}_{\mathrm{TM}}$ is undecidable.

Proof

Proof.

Suppose that we had a decider **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 **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.
3. Regularity language:
   $\mathbf{Regular}_{\mathrm{TM}} = \left\{ \langle M \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } L(M) \text{ is regular} \right\}.$
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 $w \in \mathrm{A}_{\mathrm{TM}}$) into a problem about whether some TM accepts a regular set of strings.

Proof continued...

1. Given M and w, consider the following TM 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 $L(M'_w) = \{a^n b^n \mid n \geq 0\}$.

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-A_{TM} $(\langle M, w\rangle)$
   $\quad\langle M'_w\rangle \leftarrow$ **EmbedRegularString** $(\langle M, w\rangle)$
   $\quad r \leftarrow$ **DeciderRegL**$(\langle M'_w\rangle)$.
   $\quad$**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 $\implies$ $L(M'_w) = a^n b^n \implies M$ does not accept $w \implies$ **YetAnotherDecider-A_{TM}** should reject $\langle M, w\rangle$.

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 A_{TM} as follows.

YetAnotherDecider-A_{TM} $(\langle M, w\rangle)$
$\quad\langle M'_w\rangle \leftarrow$ **EmbedRegularString** $(\langle M, w\rangle)$
$\quad r \leftarrow$ **DeciderRegL**$(\langle M'_w\rangle)$.
$\quad$**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 $\implies$ $L(M'_w) = a^n b^n \implies M$ does not accept $w \implies$ **YetAnotherDecider-A_{TM}** should reject $\langle M, w\rangle$.

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 A_{TM} as follows.

YetAnotherDecider-A_{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 $\implies L(M'_w) = a^n b^n \implies M$ does not accept $w \implies$ **YetAnotherDecider-A_{TM}** should reject $\langle M, w\rangle$.

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 A_{TM} as follows.

YetAnotherDecider-A_{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 $\implies$ $L(M'_w) = a^n b^n$ $\implies$ M 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 $\mathbf{L}$ *is a language of Turing machines; that is, each word in* $\mathbf{L}$ *encodes a* TM. *Furthermore, assume that the following two properties hold.*

(a) *Membership in* $\mathbf{L}$ *depends only on the Turing machine's language, i.e. if* $L(M) = L(N)$ *then* $\langle M \rangle \in \mathbf{L} \Leftrightarrow \langle N \rangle \in \mathbf{L}$.

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

Then $\mathbf{L}$ *is a undecidable.*