Pre-lecture brain teaser

We know that SAT is NP-complete which means that it is in NP-Hard. HALT is also in NP-Hard. Is SAT reducible to HALT? How?

- Construct a Turing machine that considers all possible assignments. Using for loops.
- If satisfying assignment is solved then halt.

Clearly oracle for HALT can find if the following Turing machine halts and therefore if the CNF is satisfiable. Is this ok? The turing machine runs in exponential time?
Pre-lecture brain teaser

We know that SAT is NP-complete which means that it is in NP-Hard. HALT is also in NP-Hard. Is SAT reducible to HALT? How?

- Construct a Turing machine that considers all possible assignments. Using for loops.
- If satisfying assignment is solved then halt.

Clearly oracle for HALT can find if the following Turing machine halts and therefore if the CNF is satisfiable.

Is this ok? The turing machine runs in exponential time?
Reductions
Reduction

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

**Definition**

Oracle $\text{ORAC}$ for language $L$ is a function that receives as a word $w$, returns $\text{TRUE} \iff w \in L$. 
Reduction

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

**Definition**

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

**Lemma**

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

- \textbf{Y}: Problem/language for which we want to prove undecidable.
Reduction proof technique

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

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

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

Create a decider for known undecidable problem $X$ using $M$.

Result in decider for $X$ (i.e., $A_{TM}$).

Contradiction $X$ is not decidable.

Thus, $L$ must be not decidable.
Reduction proof technique

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

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

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

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

Proof.
Let $T$ be a decider for $Y$ (i.e., a program or a TM). Since $X$ reduces to $Y$, it follows that there is a procedure $T_{X|Y}$ (i.e., decider) for $X$ that uses an oracle for $Y$ as a subroutine. We replace the calls to this oracle in $T_{X|Y}$ by calls to $T$. The resulting program $T_X$ is a decider and its language is $X$. Thus $X$ is decidable (or more formally TM decidable). $\square$
Lemma

Let $X$ and $Y$ be two languages, and assume that $X \implies Y$. If $X$ is undecidable then $Y$ is undecidable.
Halting
The halting problem

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

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

Similar to language already known to be undecidable:

$$A_{\text{TM}} = \left\{ \langle M, w \rangle \mid M \text{ is a } \text{TM and } M \text{ accepts } w \right\}.$$
Lemma
The language $A_{TM}$ reduces to $A_{Halt}$. Namely, given an oracle for $A_{Halt}$ one can build a decider (that uses this oracle) for $A_{TM}$. 
One way to proving that Halting is undecidable...

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

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

\begin{center}
\begin{algorithm}
\textbf{AnotherDecider-A\textsubscript{TM}}\left(\langle M, w \rangle \right)

\begin{algorithmic}
\State \texttt{res} ← \texttt{ORAC\textsubscript{Halt}}\left(\langle M, w \rangle \right)
\Comment{if $M$ does not halt on $w$ then reject.}
\If{$\texttt{res} = \texttt{reject}$}
\State {halt and reject.}
\Comment{$M$ halts on $w$ since $\texttt{res} = \texttt{accept}$.}
\Comment{Simulating $M$ on $w$ terminates in finite time.}
\State $\texttt{res}_2$ ← Simulate $M$ on $w$.
\State \Return $\texttt{res}_2$.
\end{algorithmic}
\end{algorithm}
\end{center}

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

**Theorem**

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

**Proof.**

Assume, for the sake of contradiction, that $A_{\text{Halt}}$ is decidable. As such, there is a $TM$, 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 that one can build a decider for $A_{TM}$. However, $A_{TM}$ is undecidable. A contradiction. It must be that $A_{\text{Halt}}$ is undecidable.

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

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

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

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

- Given program $\langle M \rangle$ and input $w$...
- ...can output a program $\langle M_w \rangle$.
- The program $M_w$ simulates $M$ on $w$. And accepts/rejects accordingly.
- **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 $w$. 


Embedding strings...

- Given program $\langle M \rangle$ and input $w$...
- ...can output a program $\langle M_w \rangle$.
- The program $M_w$ simulates $M$ on $w$. And accepts/rejects accordingly.
- **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 $w$. 
Theorem

The language $E_{TM}$ is undecidable.

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

```
AnotherDecider-A_{TM}(⟨M, w⟩)
⟨M_w⟩ ← EmbedString (⟨M, w⟩)
r ← TM_{ETM}(⟨M_w⟩).
if r = accept then
    return reject
// TM_{ETM}(⟨M_w⟩) rejected its input
return accept
```
Emptiness is undecidable...

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

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

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

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

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

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

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

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

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

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

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

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

**Lemma**

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

Let’s try something different. We know $E_{TM}$ is undecidable. Let’s use that:
Equality is undecidable

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

**Lemma**

The language $EQ_{TM}$ is undecidable.

Let’s try something different. We know $E_{TM}$ is undecidable. Let’s use that:

$$E_{TM} \implies EQ_{TM}$$
Equality diagram
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 \textbf{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.
DFAs
DFAs are empty?

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

What does the above language describe?

All the DFA encodings that edescribe empty languages.
DFAs are empty?

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

Is the language above decidable? Yes of course. It’s a simple DFA.
DFAs are empty?

$$E_{DFA} = \left\{ \langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset \right\}.$$  

Is the language above decidable? Yes of course. It’s a simple DFA.

**Lemma**  
*The language $E_{DFA}$ is decidable:*
Proof.
Unlike in the previous cases, we can directly build a decider (DeciderEmptyDFA) for $E_{DFA}$

**TM $R$:**

1. Input = $\langle A \rangle$
2. Mark start state of $A$ as visited.
3. Repeat until no new states get marked:
   - Mark any state that has a transition coming into it from any state that is already marked.
4. If no accept state is marked, then accept.
5. Otherwise, then reject.
Equal DFAs
DFAs are equal?

\[ EQ_{DFA} = \left\{ \langle A, b \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \right\}. \]

What does the above language describe?

All the DFA string pairs that represent equivalent languages
DFAs are equal?

$$EQ_{DFA} = \left\{ \langle A, b \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \right\}.$$

Is the language above decidable? Yes of course. Typically when we’re dealing with simple machines, they’re fairly decidable
DFAs are equal?

\[ EQ_{DFA} = \left\{ \langle A, b \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \right\} . \]

Is the language above decidable? Yes of course. Typically when we’re dealing with simple machines, they’re fairly decidable

**Lemma**

*The language* \( E_{DFA} \) *is decidable.*
DFAs are equal?

\[ EQ_{DFA} = \left\{ \langle A, b \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \right\}. \]

Is the language above decidable? Yes of course. Typically when we’re dealing with simple machines, they’re fairly decidable

**Lemma**

*The language \( E_{DFA} \) is decidable.*

Can we show this using reductions? \( EQ_{DFA} \implies E_{DFA} \)
Equal DFA trick I

Need a way to determine if there any strings in one language and not the other....
Need a way to determine if there any strings in one language and not the other....

This is known as the symmetric difference. Can create a new DFA \( C \) which represents the symmetric difference of \( L_A \) and \( L_B \).

\[
L(C) = \left( L(A) \cap \overline{L(B)} \right) \cup \left( \overline{L(A)} \cap L(B) \right)
\]

(1)
Notice with $L(C)$:

- If $L(A) = L(B)$ then $L(C) = \emptyset$
- If $L(A) \neq L(B)$ then $L(C)$ is not empty

Good time to use $E_{DFA}$ proof from before.....How do we show $EQ_{DFA}$ is decidable using a reduction?
Notice with $L(C)$:

- If $L(A) = L(B)$ then $L(C) = \emptyset$
- If $L(A) \neq L(B)$ then $L(C)$ is not empty

Good time to use $E_{DFA}$ proof from before.....How do we show $EQ_{DFA}$ is decidable using a reduction?

Want to show $EQ_{DFA} \implies E_{DFA}$
Notice with $L(C)$:

- If $L(A) = L(B)$ then $L(C) = \emptyset$
- If $L(A) \neq L(B)$ then $L(C)$ is not empty

Good time to use $E_{DFA}$ proof from before.....How do we show $EQ_{DFA}$ is decidable using a reduction?

Want to show $EQ_{DFA} \implies E_{DFA}$
Equal DFA trick II

Notice with $L(C)$:

- If $L(A) = L(B)$ then $L(C) = \emptyset$
- If $L(A) \neq L(B)$ then $L(C)$ is not empty

Good time to use $E_{DFA}$ proof from before.....How do we show $EQ_{DFA}$ is decidable using a reduction?

Want to show $EQ_{DFA} \implies E_{DFA}$
Equal DFA decider

**TM F:**

1. Input = \( \langle A, B \rangle \) where \( A \) and \( B \) are DFAs
2. Construct DFA \( C \) as described before
3. Run \texttt{DeciderEmptyDFA} from previous slide on \( C \)
4. If accepts, then accept.
5. If rejects, then reject.
Regularity
Many undecidable languages

- Almost any property defining a TM language induces a language which is undecidable.
- Proofs all have the same basic pattern.
- Regularity language:
  \[ \text{Regular}_{TM} = \left\{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular} \right\} . \]
- **DeciderRegL**: Assume TM decider for \( \text{Regular}_{TM} \).
- 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.
Outline of IsRegular? reduction

\[ \langle M, x \rangle \xrightarrow{Decider_{ATM}} \langle M_x \rangle \xrightarrow{ORAC_{RegLTM}} \]

- accept
- reject
- accept
- reject

Embed Regular String

28
Proof continued...

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

  \begin{itemize}
  \item (i) Input = $x$
  \item (ii) If $x$ has the form $a^n b^n$, halt and accept.
  \item (iii) Otherwise, simulate $M$ on $w$.
  \item (iv) If the simulation accepts, then accept.
  \item (v) If the simulation rejects, then reject.
  \end{itemize}

- not executing $M'_w$!

- feed string $\langle M'_w \rangle$ into DeciderRegL

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

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

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


Proof continued...

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

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

\[
\text{return } r
\]

- If \textbf{DeciderRegL} accepts \(\iff\) \(L(M'_w)\) regular (its \(\Sigma^*\))
• $a^n b^n$ is not regular...

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

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

• A decider for $A_{TM}$ as follows.

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

return $r$

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

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

\[
\text{AnotherDecider-\mathbb{A}_{TM}^n}(\langle M, w \rangle)
\]

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

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

\[
\text{return } r
\]

- If $\text{DeciderRegL}$ accepts $\implies L(M'_w)$ regular (its $\Sigma^*$) $\implies M$ accepts $w$. So $\text{AnotherDecider-\mathbb{A}_{TM}^n}$ should accept $\langle M, w \rangle$.
- If $\text{DeciderRegL}$ rejects $\implies L(M'_w)$ is not regular $\implies L(M'_w) = a^n b^n$
• $a^n b^n$ is not regular...

• Use DeciderRegL on $M'_w$ to distinguish these two cases.

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

• A decider for $A_{TM}$ as follows.

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

return $r$

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

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


The above proofs were somewhat repetitious...

...they imply a more general result.

**Theorem (Rice’s Theorem.)**

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

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

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

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