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**? **YES!**

**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?
SAT $\Rightarrow$ HALT

**REDUCTION:**
Construct a Turing machine $M$ that tries all assignments and halts on $\phi = 1$

$\langle \phi \rangle$ $\rightarrow$ $\langle M \rangle$ $\rightarrow$ ORAC

- ORAC HALT
  - $M$ halts
    - $\phi$ is satisfiable
    - accept
  - $M$ doesn't halt
    - $\phi$ is not satisfiable
    - reject

**DECIDER SAT**
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
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 $X \implies Y$.

\[ X \implies Y : \quad X \leq Y \]
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 ORAC for language $L$ is a function that receives as a word $w$, returns $\text{TRUE} \iff w \in L$. 

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

**Lemma**

A language *X* reduces to a language *Y*, if one can construct a TM decider for *X* using a given oracle ORAC$_Y$ for *Y*.

We will denote this fact by *X* $\implies$ *Y*.
Reduction proof technique

- **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**.

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.
• \( Y \): 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).
The contrapositive...

**Lemma**

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

If $X \implies Y$ then:

- $Y$ is decidable $\Rightarrow$ $X$ is decidable
- $X$ is undecidable $\Rightarrow$ $Y$ is undecidable
Halting
The halting problem

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

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

Similar to language already known to be undecidable:

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

Undecidable (we know!)

($\star$) $A_{\text{TM}} \Rightarrow A_{\text{Halt}}$ (we want this to prove the undecidability of $A_{\text{Halt}}$)
One way to proving that Halting is undecidable...

**Lemma**

\[ A_{TM} \Rightarrow A_{Halt} \]

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 $A_{TM}$.

(Given) $ORAC_{A_{Halt}}$:
- accept: if $M$ accepts $w$ or $M$ rejects $w$
- reject: if $M$ loops forever

(Needed) $DECIDER_{A_{TM}}$:
- accept: if $M$ accepts $w$
- reject: otherwise: if $M$ rejects $w$ or loops forever

Diagram:
- $(M, w)$ is fed into $ORAC_{A_{Halt}}$
- $ORAC_{A_{Halt}}$ runs $M$ on $w$
- If $M$ accepts or rejects, it directs to accept or reject
- If $M$ loops forever, it rejects
- The decider $DECIDER_{A_{TM}}$ accepts or rejects accordingly.
One way to proving that Halting is undecidable...

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

```
AnotherDecider-$A_{TM}(\langle M, w \rangle)$
res ← $ORAC_{Halt}(\langle M, w \rangle)$
// 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.
res₂ ← Simulate $M$ on $w$.
return res₂.
```

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

**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.
The same proof by figure...

\[ \neg \langle M, w \rangle \rightarrow \langle M, w \rangle \rightarrow ORAC_{A_{\text{HALT}}} \rightarrow \text{Simulate } M(w) \rightarrow \text{accept} \rightarrow \text{accept} \rightarrow \text{reject} \rightarrow \text{reject} \rightarrow \]

... if \( A_{\text{Hal}} \) is decidable, then \( A_{TM} \) is decidable, which is impossible.
Emptiness
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.

We want to show that \( E_{TM} \) is undecidable!

How? \( A_{TM} \leq E_{TM} \)

or \( A_{Halts} \leq E_{TM} \)
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.
Reduction:

\[ \langle M, w \rangle \quad \xrightarrow{\text{Embed } 'w'} \quad \langle Mw \rangle : \quad \langle \overline{10a_1 \ldots, 101 \ldots} \rangle \quad \rightarrow \quad \langle \overline{1001 \ldots 101 \ldots} \rangle \]

E.g. \( f(x) : \)

\[
\begin{align*}
w &= 2 \\
\text{return } w^2
\end{align*}
\]

\( f(2) = 4 \)

\( f(8) = 4 \)

Reduction

\[ \langle M, w \rangle \quad \xrightarrow{\text{give accept}} \quad \text{if } M \text{ accepts } w \]

\[ \langle M, w \rangle \quad \xrightarrow{\text{give reject}} \quad \text{if } M \text{ rejects } w \]

\[ L(Mw) = \Sigma^* : L \text{ is non-empty} \]

\[ L(Mw) = \emptyset : L \text{ is empty} \]
The language of empty languages

- $E_{\text{TM}} = \left\{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset \right\}$.
- $TM_{ETM}$: Assume we are given this decider for $E_{\text{TM}}$.
- Need to use $TM_{ETM}$ to build a decider for $A_{\text{TM}}$.
- Decider for $A_{\text{TM}}$ 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.
• 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 $\uparrow\downarrow$ or $\downarrow$; if $M$ accepts $w$, and otherwise $\uparrow$.
• 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 $\text{AnotherDecider-}A_{TM}$ for $A_{TM}$:

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

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

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

if $r = \text{accept}$ then

return reject

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

return accept
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 $\langle M, w \rangle$. 

Emptiness 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 $\langle M, w \rangle$.

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

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

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 \(\langle M, w \rangle\).

\[\implies\text{AnotherDecider-}A_{TM}\text{ is decider for } A_{TM}\]

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

...must be assumption that \(E_{TM}\) 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.
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

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

**Lemma**

The language \text{EQ}_{TM} is undecidable.

Let’s try something different. We know \text{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:

\[ E_{TM} \iff EQ_{TM} \]
Reduction:

\[ <M> \rightarrow <M>, <N> \]

Idea: Take \( N \) to be a \( TM \) such that \( L(N) = \phi \).

\[ <M> \rightarrow <M>, <N> \] \( (L(N) = \phi) \) \rightarrow \text{ORAC} \quad \text{EQ}_{TM} \]

\[ \text{accept if } L(M) = L(N) = \phi \]

\[ \text{reject if } L(M) \neq L(N) = \phi \]
Proof

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

\textbf{TM} $R$:

1. Input = $\langle M \rangle$
2. Include the (constant) code for a TM $T$ that rejects all its input. We denote the string encoding $T$ by $\langle T \rangle$.
3. Run \textbf{DeciderEqual} on $\langle M, T \rangle$.
4. If \textbf{DeciderEqual} accepts, then accept.
5. If \textbf{DeciderEqual} rejects, then reject.
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 describe 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:

We can make a decider for \( E_{DFA} \). The decider is a TM that simulates the working of \( A \) and checks if \( L(A) = \emptyset \). Simple DFA computation. See page 22 for the formal description of the decider.
Proof.
Unlike in the previous cases, we can directly build a decider \( \text{DeciderEmptyDFA} \) 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.
DFA 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 \( EQ_{DFA} \) is decidable.

Can we show this using reductions?

Using an ORACLE for \( E_{DFA} \), how?

Decidable! (Just saw.)

We want to show that this is decidable.

next page.
Need a way to determine if there any strings in one language and not the other....
Equal DFA trick I

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 $L_B$.

$L(C) = \left( L(A) \cap \overline{L(B)} \right) \cup \left( \overline{L(A)} \cap L(B) \right)$  \hspace{1cm} (1)
$E_{Q_{DFA}} \Rightarrow E_{DFA}$

Reduction:
Construct $C$ such that:
$L(C) = \Delta(L(A), L(B))$

ORAC $E_{DFA}$

accept

reject

$L(C) = \emptyset$

$L(A) = L(B)$

$L(C) \neq \emptyset$
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 $EQ_{DFA} \iff E_{DFA}$
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 **DeciderEmptyDFA** from previous slide on \( C \)
4. If accepts, then accept.
5. If rejects, then reject.
Regularity (RIY)
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:
  \[
  \text{Regular}_{TM} = \left\{ \langle M \rangle \mid M \text{ is a } TM \text{ and } L(M) \text{ is regular} \right\}.
  \]
- **DeciderRegL**: Assume TM decider for 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

\( A_{TM} \Rightarrow Reg_{LTM} \)

\( \langle M, w \rangle \rightarrow \langle M_x^w \rangle \rightarrow ORAC_{RegLTM} \rightarrow \) accept/reject

\( M \) accepts \( w \) then \( L(M) = \Sigma^* : \text{reg.} \)

\( M \) doesn't accept \( w \) then \( + x \in \Sigma^* : n \geq 0 \) is accepted by \( M \).

\( \Rightarrow L(M) \) has an infinite fooling set.

\( \Rightarrow \text{Non-Reg.} \)

\( M \) accepts \( w \) then any \( x \) is accepted by \( M \)

\( \Rightarrow L(M) = \Sigma^* : \text{reg.} \)

only

\( M \) doesn't accept \( w \) then \( x \in \Sigma^* b^n : n \geq 0 \)

is accepted by \( M \).

\( \Rightarrow L(M) \) has an infinite fooling set.

\( \Rightarrow \text{Non-Reg.} \)

\( M_w(x) : \\
\text{if } x \text{ is of the form } a^n b^n \\
\text{accept} \\
\text{otherwise} \\
\text{simulate } M \text{ on } w \\
\text{if accept} \\
\text{accept} \\
\text{is reject} \\
\text{reject} \)
Given $M$ and $w$, consider the following TM $M'_w$:

$\text{TM } M'_w$:  
(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 
- **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 $\text{DeciderRegL}$ on $M'_w$ to distinguish these two cases.

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

• A decider for $A_{TM}$ as follows.

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

\begin{center}
\text{return } r
\end{center}

• If $\text{DeciderRegL}$ accepts $\implies L(M'_w)$ regular (its $\Sigma^*$)
• $a^n b^n$ 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.

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

\[
\text{AnotherDecider-} A_{TM}(\langle M, w \rangle)
\]

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

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

return \(r\)

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

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

30
Rice’s theorem

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.
Philosophical Question:

What can we do to decide about languages accepted by Turing machines / programs?

"Nothing" except "trivial" things!