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?
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?
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 $X \iff Y$. 
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 $X \Longrightarrow Y$.

Definition

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

• **Y**: Problem/language for which we want to prove undecidable.
• \textbf{Y}: Problem/language for which we want to prove undecidable.

• Proof via reduction. Result in a proof by contradiction.
• **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**.
• **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).
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 \( w \):

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

Similar to language already known to be undecidable:

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

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

\[
\text{AnotherDecider}_{A_{\text{TM}}} (\langle M, w \rangle) \\
res \leftarrow \text{ORAC}_{\text{Halt}} (\langle M, w \rangle) \\
// \text{ if } M \text{ does not halt on } w \text{ then reject.} \\
\text{if } res = \text{ reject then} \\
\quad \text{halt and reject.} \\
// M \text{ halts on } w \text{ since } res = \text{accept.} \\
// \text{ Simulating } M \text{ on } w \text{ terminates in finite time.} \\
res_2 \leftarrow \text{Simulate } M \text{ on } w. \\
\text{return } res_2.
\]

This procedure always return and as such its a decider for \( A_{\text{TM}} \).
**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...

... 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

- \( E_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset \} \).
- \( TM_{ETM} \): 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.
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 \( E_{TM} \).

\[ \text{Need to use } TM_{ETM} \text{ to build a decider for } A_{TM}. \]

\[ \text{Decider for } A_{TM} \text{ is given } M \text{ and } w \text{ and must decide whether } M \text{ accepts } w. \]

\[ \text{Restructure question to be about Turing machine having an empty language.} \]

\[ \text{Somehow make the second input (} w \text{) disappear.} \]

\[ \text{Idea: hard-code } w \text{ into } M, \text{ creating a } TM \ M_w \text{ which runs } M \text{ on the fixed string } w. \]

\[ TM \ M_w(x): \]
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.
• $\text{EmbedString}(\langle M, w \rangle)$ input two strings $\langle M \rangle$ and $w$, and output a string encoding $(\text{TM}) \langle M_w \rangle$.
• 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 $A_{TM}$:

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

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

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

if $r = \text{accept}$ then
   return reject

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

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

\( \implies \) \textit{AnotherDecider-} \( A_{TM} \) 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

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

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

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

**Lemma**

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

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

\[ E_{TM} \implies EQ_{TM} \quad \]
Proof.
Suppose that we had a decider \texttt{DeciderEqual} for \textit{EQ}_{TM}. Then we can build a decider for \textit{E}_{TM} as follows:

\textbf{TM \textit{R}:}

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

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

What does the above language describe?
$E_{\text{DFA}} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset \}$. 

Is the language above decidable?
DFAs are empty?

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

Is the language above decidable?

**Lemma**

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

\textbf{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?

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

What does the above language describe?
DFAs are equal?

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

Is the language above decidable?
DFAs are equal?

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

Is the language above decidable?

**Lemma**

The language \( E_{\text{DFA}} \) 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?

**Lemma**

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

Can we show this using reductions?
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 \( L_B \).

\[
L(C) = \left( L(A) \cap \overline{L(B)} \right) \cup \left( \overline{L(A)} \cap L(B) \right)
\]
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?
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 $EQ_{DFA} \implies E_{DFA}$
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 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:
  \[
  \text{Regular}_{TM} = \left\{ \langle M \rangle \middle| M \text{ is a TM and } L(M) \text{ is regular} \right\}.
  \]
- **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

Diagram: 

- $\langle M, x \rangle$ 
  - $\text{Embed Regular String}$ 
  - $\langle M_x \rangle$ 
  - $\text{Decider}_{ATM}$ 

- $\text{ORAC}_{RegLTM}$ 
  - accept 
  - reject 

- accept 
- 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 $\text{DeciderRegL}$
- $\textbf{EmbedRegularString}$: 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 $L(M'_w) = \{a^n b^n \mid n \geq 0\}$.
• $a^n b^n$ 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 $A_{\text{TM}}$ as follows.

```
AnotherDecider-\text{A}_{\text{TM}}(\langle M, w \rangle)
\begin{align*}
\langle M'_w \rangle & \leftarrow \text{EmbedRegularString}(\langle M, w \rangle) \\
               & \quad r \leftarrow \text{DeciderRegL}(\langle M'_w \rangle).
\end{align*}
```

return $r$

• If $\text{DeciderRegL}$ accepts $\implies L(M'_w)$ regular (its $\Sigma^*$)
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 \(\implies L(M'_w)\) regular (its \(\Sigma^*\)) \(\implies M\) accepts \(w\). So \textbf{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 $A_{TM}$ as follows.

```plaintext
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 $\langle M, w \rangle$.
- If $\text{DeciderRegL}$ 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.

```
AnotherDecider-ATM(⟨M, w⟩)
⟨M'⟩ ← EmbedRegularString(⟨M, w⟩)
r ← DeciderRegL(⟨M'_w⟩).
return r
```

- If DeciderRegL accepts $\implies L(M'_w)$ regular (its $\Sigma^*$) $\implies M$ accepts $w$. So AnotherDecider-ATM should accept $⟨M, w⟩$.
- 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-ATM should reject $⟨M, w⟩$. 
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 undecidable.