Some Turing machines never go into an infinite loop; they always halt. These are called \textit{deciders}.

For deciders, accepting is the same as not rejecting and rejecting is the same as not accepting:

\begin{align*}
\text{Does not reject} & \quad \text{Accept} \quad \text{Halts (always)} \\
\text{Does not accept} & \quad \text{Reject}
\end{align*}

\textbf{Decidable languages}

A language is called \textit{Turing-decidable} (or just \textit{decidable}) iff there is a decider $M$ such that $L(M) = L$.

Given a decider $M$, you can learn whether or not a string $w \in L(M)$ by running $M$ on $w$; it will eventually accept or reject $w$.

The set $R$ is the set of all Turing-decidable languages.

\[ L \in R \iff L \text{ is Turing-decidable.} \]
**R and RE languages**

Intuitively, a language is in **RE** if there is some way that you could exhaustively search for a proof that $w \in L$.

- If you find it, accept!
- If you don’t find one, keep looking!

Intuitively, a language is in **R** if there is a concrete algorithm that can determine whether $w \in L$.

It tends to be *much harder* to show that a language is in **R** than in **RE**.

---

**$A_{TM}$ and HALT**

Both $A_{TM}$ and HALT are undecidable.

There is no way to decide whether a TM will accept or eventually terminate.

However, both $A_{TM}$ and HALT are recognizable.

We can always run a TM on a string $w$ and accept if that TM accepts or halts.

*Intuition:* The only general way to learn what a TM will do on a given string is to run it and see what happens.

---

**RE and co-RE**

A language $L$ is in **RE** iff there is a TM $M$ such that

- if $w \in L$, then $M$ accepts $w$
- if $w \notin L$, then $M$ does not accept $w$

A TM $M$ of this sort is called a *recognizer*, and $L$ is called *Turing-recognizable*.

A language $L$ is in *co-RE* iff there is a TM such that

- if $w \in L$, then $M$ does not reject $w$
- if $w \notin L$, then $M$ rejects $w$

A TM $M$ of this sort is called a *co-recognizer* and the language $L$ is called *co-Turing-recognizable*. 
There is a TM $M$ where $M$ rejects $w$ iff $w \not\in L_{RE}$ and $co\cdot RE$.

There is a TM $M$ where $M$ accepts $w$ iff $w \in L_{RE}$.

Proof by reduction

A simple reduction

I wonder if I can lift this car…
A simple reduction

Ow! No, I can’t!

A simple reduction

I wonder if I can lift this fully loaded truck…

A simple reduction

Nope! If I could lift this fully loaded truck, I’d be able to lift the car!

A reduction is a way of solving a problem easy with a problem harder.

In this example, we could solve the problem of “lift the car” by reducing it to the harder problem of “lift the truck”.

harder can be used to solve easy
A reduction works by turning an instance of *easy* into an instance of *harder*.

In this example, we reduce the problem of lifting the car by putting it into a truck and lifting the truck.

Suppose we can’t solve *easy*.

If we can reduce *easy* to *harder*, we cannot solve *harder* either.

Suppose we can’t solve *easy*.

If we can reduce *easy* to *harder*, we cannot solve *harder* either.
Suppose we can’t solve *easy*.

If we can *reduce* *easy* to *harder*, we cannot solve *harder* either.

Reductions and decidability

Suppose we want to prove that some problem *harder* is undecidable.

If we can *reduce* an undecidable problem *hard* to *harder*, then we know that we cannot decide *harder*.
Reductions and decidability

Suppose we want to prove that some problem *harder* is undecidable.

If we can reduce an undecidable problem *hard* to *harder*, then we know that we cannot decide *harder*.

Proof by reduction

Suppose that we are given a language $L$ that we believe is undecidable.

We can prove that this is true using the following technique:

Assume, for the sake of contradiction, that $L$ is decidable.

Show how a decider for $L$ could be used to construct a decider for an undecidable language.

Conclude that $L$ must not be decidable.

Reduction from the Halting Problem

The language $HALT_{TM} \in \text{RE}$ (i.e., is Turing-recognizable), but is not Turing-decidable:

$HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ halts on } w \}$

If we can reduce $HALT_{TM}$ to some problem $P$, then that new problem cannot be decidable.
Example reduction: The Totality Problem

Is there a way to tell whether a given Turing machine is a recognizer or a decider?

I.e., does the Turing machine halt on all inputs?

This is called the totality problem.
Let \( TOTAL = \{ \langle M \rangle \mid M \text{ halts on all inputs} \} \).

This seems at least as hard as the Halting Problem. How could we prove that it’s undecidable?

**Proof idea:** Reduce \( \text{HALT}_{TM} \) to \( TOTAL \).

Show how a decider for \( TOTAL \) could be used to build a decider for \( \text{HALT}_{TM} \).

Conclude that no such decider can exist.
We're going to try to show how a decider for \textsc{Total} decides \textsc{Halt}_{TM}, so our input must be a TM and a string.

Construct $M'$ from $\langle M, w \rangle$

$M' = \text{"On input } x:\$
1. Ignore $x$.
2. Run $M$ on $w$.
3. If $M$ accepts $w$, accept.
4. If $M$ rejects $w$, reject.
The Totality Problem

Construct $M'$ from $\langle M, w \rangle$

Decider for $\text{TOTAL}$

$\langle M, w \rangle$

$x$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$

$\langle M', w \rangle$
The Totality Problem

Construct \( M' \) from \( \langle M, w \rangle \) (Always halts)

Simulate \( M \) on \( w \)

(ignored)

Decider for TOTAL

What does \( D \) do if \( M \) halts on \( w \)?

(Always halts)

The Totality Problem

Construct \( M' \) from \( \langle M, w \rangle \) (Never halts)

Simulate \( M \) on \( w \)

(ignored)

Decider for TOTAL

What does \( D \) do if \( M \) loops on \( w \)?

(Never halts)

The Totality Problem

Construct \( M' \) from \( \langle M, w \rangle \) (Never halts)

Simulate \( M \) on \( w \)

(ignored)

Decider for TOTAL

What does \( D \) do if \( M \) loops on \( w \)?

(Never halts)

The Totality Problem

Construct \( M' \) from \( \langle M, w \rangle \) (Always halts)

Simulate \( M \) on \( w \)

(ignored)

Decider for TOTAL

What does \( D \) do if \( M \) halts on \( w \)?

(Always halts)

D is a decider for HALT_{TM}!
What just happened?
Let’s walk through this construction in detail.

Suppose, for the sake of contradiction, that TOTAL is decidable.
Build a TM $D$ that accepts $\langle M, w \rangle$ and constructs a TM $M'$ that is a decider iff $M$ accepts $w$.

This is the key step in most reductions. We build a TM that has a property of the new problem (here, TOTAL) based on whether some other TM has a property of the old problem (here, HALT$_{TM}$).

Deciding whether this TM has the new property thus decides whether some other TM has the old property.

Theorem: TOTAL is undecidable.
Proof: By contradiction; assume that TOTAL is decidable. Let $T$ be a decider for TOTAL. Then consider the following TM:

$D =$ “On input $\langle M, w \rangle$:
1. Construct the TM $M' = 'On input x:
   1. Ignore x.
   2. Run $M$ on $w$.
   3. If $M$ accepts $w$, accept.
   4. If $M$ rejects $w$, reject.’

Most reduction proofs work by building a new TM out of an existing TM and a string. The new TM then has some property iff the old TM/string pair has some property.

The behavior of $D$ depends on what $T$ does on $\langle M' \rangle$.
The behavior of $T$ on $\langle M' \rangle$ depends on what $M$ does on $w$.
So the behavior of TM $D$ depends on what $M$ does on $w$.

What just happened?
Let’s walk through this construction in detail.

Suppose, for the sake of contradiction, that TOTAL is decidable.
Build a TM $D$ that accepts $\langle M, w \rangle$ and constructs a TM $M'$ that is a decider iff $M$ accepts $w$.

Using the decider for TOTAL, check whether $M'$ is a decider:
If $M'$ is a decider, then $M$ halts on $w$.
If $M'$ is not a decider, then $M$ does not halt on $w$.
Conclude that if TOTAL is decidable, then HALT$_{TM}$ is decidable.
Since we know that HALT$_{TM}$ is undecidable, our assumption was wrong, and TOTAL is undecidable.
Theorem: TOTAL is undecidable.

Proof: By contradiction; assume that TOTAL is decidable. Let T be a decider for TOTAL. Then consider the following TM:

\[ D = \text{"On input } \langle M, w \rangle:\]
1. Construct the TM \( M' = \text{On input } x:\)
   1. Ignore \( x.\)
   2. Run \( M \) on \( w.\)
   3. If \( M \) accepts \( w, \text{accept.}\)
   4. If \( M \) rejects \( w, \text{reject.}\)
2. Run \( T \) on \( \langle M' \rangle.\)
3. If \( T \) accepts, \( \text{accept.}\)
4. If \( T \) rejects, \( \text{reject.}\)

We claim that \( D \) decides \( \text{HALT}_{TM}. \) To see this, we show that \( D \) is a decider and that \( L(D) = \text{HALT}_{TM}. \) To see that \( D \) is a decider, note that after we construct \( M', \) we run \( T \) on \( \langle M' \rangle. \) Since \( T \) is a decider, it always halts, so \( D \) always halts.

Example reduction: \( \text{REGULAR}_{TM} \)

To see that \( L(D) = \text{HALT}_{TM}, \) note that \( D \) accepts \( \langle M, w \rangle \) iff \( T \) accepts \( \langle M' \rangle. \) Because \( T \) is a decider for TOTAL, \( T \) accepts \( \langle M' \rangle \) iff \( M' \) halts on all inputs. By construction, \( M' \) halts on any input iff \( M \) halts on \( w. \) Finally, \( M \) halts on \( w \) iff \( \langle M, w \rangle \in \text{HALT}_{TM}. \) This means that \( D \) accepts \( \langle M, w \rangle \) iff \( \langle M, w \rangle \in \text{HALT}_{TM}, \) so \( L(D) = \text{HALT}_{TM}. \)

We have reached a contradiction, because \( D \) decides \( \text{HALT}_{TM}, \) which we know is undecidable. Thus our assumption was wrong and TOTAL is undecidable. ■

Testing regularity

We know that every \textit{regular language} is decidable (and Turing-recognizable and context-free).

Thus some TMs must recognize regular languages.

Can we decide whether a TM recognizes a regular language?
Testing regularity

Let

\[ \text{REGULAR}_{\text{TM}} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular} \} \]

This doesn’t seem to have any obvious connection to \( \text{HALT}_{\text{TM}} \) or \( \text{A}_{\text{TM}} \).

Is \( \text{REGULAR}_{\text{TM}} \) decidable?

Unfortunately, it isn’t, which we can prove by reduction from \( \text{A}_{\text{TM}} \). (We could also prove it by reduction from \( \text{HALT}_{\text{TM}} \).)

**Theorem:** \( \text{REGULAR}_{\text{TM}} \) is undecidable.

**Proof idea:** Suppose \( \text{REGULAR}_{\text{TM}} \) is decidable by some machine \( R \).

Given a TM \( M \) and a string \( w \), construct a TM \( M_2 \) with these properties:

1. If \( M \) accepts \( w \), then \( L(M_2) \) is regular.
2. If \( M \) does not accept \( w \), then \( L(M_2) \) is not regular.

Have \( R \) decide whether or not \( M_2 \) is regular.

1. If \( M_2 \) is regular, \( M \) accepts \( w \).
2. If \( M_2 \) is not regular, \( M \) does not accept \( w \).

We can use \( R \) to decide \( \text{A}_{\text{TM}} \), which is a contradiction.
Theorem: \( \text{REGULAR}_{\text{TM}} \) is undecidable.

Proof idea: Suppose \( \text{REGULAR}_{\text{TM}} \) is decidable by some machine \( R \).

Given a TM \( M \) and a string \( w \), construct a TM \( M_2 \) with these properties:
1. If \( M \) accepts \( w \), then \( L(M_2) = \Sigma^* \).
2. If \( M \) does not accept \( w \), then \( L(M_2) \) is not regular.

Have \( R \) decide whether or not \( M_2 \) is regular.
1. If \( M_2 \) is regular, \( M \) accepts \( w \).
2. If \( M_2 \) is not regular, \( M \) does not accept \( w \).

We can use \( R \) to decide \( A_{\text{TM}} \), which is a contradiction.

Building \( M_2 \)

\[ \begin{array}{c}
M_2 \\
\hline
x \\
\hline
x = 0^n1^n?
\end{array} \]
Building $M_2$

$M_2$

$x = 0^n 1^n$?

Simulate $M$ on $w$

Building $M_2$

$M_2$

$x = 0^n 1^n$?

Simulate $M$ on $w$

Building $M_2$

$M_2$

$x = 0^n 1^n$?

Simulate $M$ on $w$

Building $M_2$

$M_2$

$x = 0^n 1^n$?

Simulate $M$ on $w$

$M_2 = \text{"On input } x:\$

1. If $x$ has the form $0^n 1^n$, accept.
2. Otherwise, run $M$ on $w$.
3. If $M$ accepts $w$, accept."

If $M$ accepts $w$, then $M_2$ accepts all strings.
If $M$ doesn’t accept $w$, then $M_2$ only accepts strings of the form $0^n 1^n$. 
**Theorem:** \( \text{REGULAR}_{TM} \) is undecidable.

**Proof:** By contradiction; assume \( R \) decides \( \text{REGULAR}_{TM} \). Consider the following machine \( S \):

\[
S = \text{"On input } \langle M, w \rangle, \text{ where } M \text{ is a TM and } w \text{ is a string:}
\]

1. Construct the machine \( M_2 = \text{"On input } x: \)
   1. If \( x \) has the form \( \emptyset^*1^n \), accept.
   2. Otherwise, run \( M \) on \( w \).
   3. If \( M \) accepts \( w \), accept.
2. Run \( R \) on input \( \langle M_2 \rangle \).
3. If \( R \) accepts, accept; if \( R \) rejects, reject.

We claim that \( S \) is a decider and that \( L(S) = A_{TM} \).

---

**We need to show both of these:**

1. \( S \) is a decider (i.e., always halts).
2. \( S \) accepts exactly the strings in \( A_{TM} \).

---

\[
S = \text{"On input } \langle M, w \rangle, \text{ where } M \text{ is a TM and } w \text{ is a string:}
\]

1. Construct the machine \( M_2 = \text{"On input } x: \)
   1. If \( x \) has the form \( \emptyset^*1^n \), accept.
   2. Otherwise, run \( M \) on \( w \).
   3. If \( M \) accepts \( w \), accept.
2. Run \( R \) on input \( \langle M_2 \rangle \).
3. If \( R \) accepts, accept; if \( R \) rejects, reject.

We claim that \( S \) is a decider and that \( L(S) = A_{TM} \).

1. To see that \( S \) is a decider:
   1. \( S \) accepts \( \langle M, w \rangle \) iff \( R \) accepts \( \langle M_2 \rangle \). Since \( R \) decides \( \text{REGULAR}_{TM} \), \( R \) accepts \( \langle M_2 \rangle \) iff \( L(M_2) \) is regular. We claim that \( L(M_2) \) is regular iff \( M \) accepts \( w \). To see this, note that if \( M \) accepts \( w \), \( M_2 \) accepts all strings, either because the string has form \( \emptyset^*1^n \) or because it accepts in the final step after \( M \) accepts \( w \). Thus \( L(M_2) = \Sigma^* \), which is regular.
   2. If \( M \) does not accept \( w \), then \( M \) only accepts \( x \) if it has the form \( \emptyset^*1^n \). Thus \( L(M_2) = \{ \emptyset^*1^n \mid n \in \mathbb{N} \} \), which is not regular. Thus \( S \) accepts \( \langle M, w \rangle \) iff \( M \) accepts \( w \) iff \( \langle M, w \rangle \in A_{TM} \), so \( L(S) = A_{TM} \).
2. To see that \( L(S) = A_{TM} \):
   1. \( S \) is a decider and that \( L(S) = A_{TM} \).
   2. To see that \( L(S) = A_{TM} \):

   We have reached a contradiction because we know that \( A_{TM} \) is undecidable. Thus our assumption was wrong and \( \text{REGULAR}_{TM} \) is undecidable. \( \blacksquare \)
Consider the following problems:

- $L_D$: Does $M$ reject $⟨M⟩$?
- $A_{TM}$: Does $M$ accept $w$?
- $HALT_{TM}$: Does $M$ halt on $w$?
- $TOTAL$: Does $M$ halt on all inputs?
- $REGULAR_{TM}$: Is $L(M)$ regular?

A property of an RE language is some trait that may apply to RE languages.

For example:

- Does $L = ∅$?
- Is $L$ regular?
- Is $L$ context-free?
- Does $L$ contain any string of length exactly 137?
We can describe a property of an RE language as the set of RE languages with that property.

If \( P \) is a property of RE languages, consider the language

\[
L_P = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) \in P \}
\]

Note that membership in \( L_P \) depends only on the language of a TM, not the description of that TM.

If \( L(M_1) = L(M_2) \), then \( \langle M_1 \rangle \in L_P \) iff \( \langle M_2 \rangle \in L_P \)

A property of RE languages is called trivial if all RE languages have the property or no RE languages have the property, e.g.,

\[
\{ \langle M \rangle \mid L(M) \text{ is RE} \} \text{ is trivial}
\]

\[
\{ \langle M \rangle \mid L(M) \text{ is not RE} \} \text{ is trivial}
\]

A property of RE languages is called nontrivial if there exist TMs \( M_1 \) and \( M_2 \) such that \( \langle M_1 \rangle \in L_P \), but \( \langle M_2 \rangle \not\in L_P \), e.g.,

\[
\{ \langle M \rangle \mid L(M) \text{ is infinite} \} \text{ is nontrivial}
\]

\[
\{ \langle M \rangle \mid L(M) \text{ is regular} \} \text{ is nontrivial}
\]

\[
\{ \langle M \rangle \mid L(M) \text{ is decidable} \} \text{ is nontrivial}
\]

\[
L_{\text{even}} = \{ \langle M \rangle \mid L(M) \text{ is finite and } |L(M)| \text{ is even} \}
\]

This is a property of RE languages, because it depends purely on the language of the TM and not on the TM itself.

Specifically, if \( L(M_1) = L(M_2) \), then \( \langle M_1 \rangle \in L_{\text{even}} \) iff \( \langle M_2 \rangle \in L_{\text{even}} \)

\[
L_{\text{evenQ}} = \{ \langle M \rangle \mid M \text{ has an even number of states} \}
\]

This is not a property of RE languages, because it does not depend purely on the language of the TM.

Specifically, if \( L(M_1) = L(M_2) \), then it may be possible for \( \langle M_1 \rangle \in L_{\text{evenQ}} \) but \( \langle M_2 \rangle \not\in L_{\text{evenQ}} \)

Rice’s Theorem

Any nontrivial property of the RE languages is undecidable.
Can we apply Rice's Theorem to this language?
$L_{ne} = \{\langle M \rangle \mid L(M) \neq \emptyset\}$

We can apply Rice's Theorem if two conditions hold:

✅ **$L_{ne}$ is nontrivial:**
\[ \exists M_1 \cdot \exists M_2 . \langle M_1 \rangle \in L_{ne} \land \langle M_2 \rangle \notin L_{ne} \]

✅ **$L_{ne}$ is a property of RE languages:**
If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in L_{ne}$ iff $\langle M_2 \rangle \in L_{ne}$

Rice's Theorem applies; $L_{ne}$ is undecidable.

Can we apply Rice's Theorem to this language?
$L_{es} = \{\langle M \rangle \mid M \text{ has an even number of states}\}$

We can apply Rice's Theorem if two conditions hold:

✅ **$L_{es}$ is nontrivial:**
\[ \exists M_1 \cdot \exists M_2 . \langle M_1 \rangle \in L_{es} \land \langle M_2 \rangle \notin L_{es} \]

❌ **$L_{es}$ is a property of RE languages:**
If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in L_{es}$ iff $\langle M_2 \rangle \in L_{es}$

Rice's Theorem does not apply.

Can we apply Rice's Theorem to this language?
$L_{small} = \{\langle M \rangle \mid \text{There is a five-state TM that recognizes } L(M)\}$

We can apply Rice's Theorem if two conditions hold:

✅ **$L_{small}$ is nontrivial:**
\[ \exists M_1 \cdot \exists M_2 . \langle M_1 \rangle \in L_{es} \land \langle M_2 \rangle \notin L_{es} \]

✅ **$L_{small}$ is a property of RE languages:**
If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in L_{small}$ iff $\langle M_2 \rangle \in L_{small}$

Rice's Theorem applies; $L_{small}$ is undecidable.

Rice’s Theorem tells us that all of the following problems are undecidable:

$L_{\text{palindrome}} = \{\langle M \rangle \mid \text{every string in } L(M) \text{ is a palindrome}\}$

$L_{\text{allodd}} = \{\langle M \rangle \mid \text{every string in } L(M) \text{ has odd length}\}$

$L_{\text{CFL}} = \{\langle M \rangle \mid L(M) \text{ is a context-free language}\}$

$L_{\text{short}} = \{\langle M \rangle \mid L(M) \text{ has no strings of length greater than 5}\}$

$L_{\text{decidable}} = \{\langle M \rangle \mid L(M) \text{ is decidable}\}$

$E_{\text{TM}} = \{\langle M \rangle \mid L(M) = \emptyset\}$
Appendix: Proving Rice’s theorem

The proof of Rice’s Theorem is a generalization of the reductions we’ve seen so far.

**General idea:** If $L_P$ is a nontrivial property of RE languages, show that we can reduce $\text{HALT}_{TM}$ to $L_P$.

**Proof sketch:** Suppose that some nontrivial property of RE languages, $L_P$, is decidable.

Construct a TM $H$ that works as follows:

On input $\langle M, w \rangle$:
- Construct TM $M'$ such that $\langle M' \rangle \in L_P$ iff $M$ halts on $w$.
- Use the decider for $L_P$ to decide whether $\langle M' \rangle \in L_P$.

Conclude that $H$ decides $\text{HALT}_{TM}$, which is impossible.

---

**Key Idea 1**

Assume, for the sake of contradiction, that $L_P$ is a decidable, nontrivial property of RE languages.

Since $R$ is closed under complement, if we can show that either $L_P$ is undecidable or that $\overline{L_P}$ is undecidable, we will have reached a contradiction.

If $M_\emptyset$ is a TM that accepts the empty language, then either $\langle M_\emptyset \rangle \in L_P$ or $\langle M_\emptyset \rangle \notin L_P$.

- If $\langle M_\emptyset \rangle \in L_P$, we prove that $L_P$ is undecidable.
- If $\langle M_\emptyset \rangle \notin L_P$, we prove that $\overline{L_P}$ is undecidable.

---

**Key Idea 2**

Assume without loss of generality that $\langle M_\emptyset \rangle \notin L_P$.

$L_P$ is a nontrivial property of RE languages, so there must be some TM $M_{yes}$ for which $\langle M_{yes} \rangle \in L_P$.

$L_P$ is a nontrivial property of RE languages, so $L(M_{yes}) \neq L(M_\emptyset)$. 
Recognizer for $M_{yes}$

Simulate $M$ on $w$
M' = “On input x:
  Run M on w.
  If M halts, run $M_{yes}$ on x.
  Accept if $M_{yes}$ accepts x.
  Reject if $M_{yes}$ rejects x.”
Machine $M'$

$M' = "On input x:
Run $M$ on $w$.
If $M$ halts, run $M_{yes}$ on $x$.
Accept if $M_{yes}$ accepts $x$.
Reject if $M_{yes}$ rejects $x$."
Machine $M'$

$M'$ = “On input $x$:
Run $M$ on $w$.  
If $M$ halts, run $M_{\text{yes}}$ on $x$.  
Accept if $M_{\text{yes}}$ accepts $x$.  
Reject if $M_{\text{yes}}$ rejects $x$.”

If $M$ halts on $w$, what does $M'$ do on $x$?

$L(M') = L(M_{\text{yes}})$
Key Idea 3

\( M' = \text{“On input } x: \)

1. Run \( M \) on \( w \).
2. If \( M \) halts, run \( M_{\text{yes}} \) on \( x \).
3. Accept if \( M_{\text{yes}} \) accepts \( x \).
4. Reject if \( M_{\text{yes}} \) rejects \( x \)."

If \( M \) halts on \( w \), \( L(M') = L(M_{\text{yes}}) \)
If \( M \) loops on \( w \), \( L(M') = \emptyset \)

Remember that \( M_{\emptyset} \) is a machine that never accepts, so \( L(M_{\emptyset}) = \emptyset \)
Key Idea 3

$M' = \text{"On input } x:\"

1. Run $M$ on $w$.
2. If $M$ halts, run $M_{yes}$ on $x$.
3. Accept if $M_{yes}$ accepts $x$.
4. Reject if $M_{yes}$ rejects $x$.

If $M$ halts on $w$, $\langle M' \rangle \in L_P$
If $M$ loops on $w$, $L(M') = L(M_{\emptyset})$
**Key Idea 3**

\[ M' = \text{“On input } x: \]

1. Run \( M \) on \( w \).
2. If \( M \) halts, run \( M_{\text{yes}} \) on \( x \).
3. Accept if \( M_{\text{yes}} \) accepts \( x \).
4. Reject if \( M_{\text{yes}} \) rejects \( x \)."

If \( M \) halts on \( w \), \( \langle M' \rangle \in L_P \)

If \( M \) loops on \( w \), \( \langle M' \rangle \not\in L_P \)

*This is the key step of the construction. If we can decide whether \( \langle M' \rangle \in L_p \), we would be deciding whether \( M \) halts on \( w \).*
The Complete Construction

Construct $M'$ from $\langle M, w \rangle$

Recognize for $M_{yes}$

Simulate $M$ on $w$

Decide for $L_p$

Construct $M'$ from $\langle M, w \rangle$

Recognize for $M_{yes}$

Simulate $M$ on $w$

Decide for $L_p$

What happens if $M$ halts on $w$?

$L(M') \in L_p$
The Complete Construction

Simulate M on w

Recognize for $M_{yes}$

L$(M') \in L_p$

What happens if M loops on w?

Decider for $L_p$

Construct $M'$ from $(M, w)$

The Complete Construction

Simulate M on w

Recognize for $M_{yes}$

L$(M') \in L_p$

What happens if M loops on w?

Decider for $L_p$

Construct $M'$ from $(M, w)$

The Complete Construction

Simulate M on w

Recognize for $M_{yes}$

L$(M') \in L_p$

What happens if M loops on w?

Decider for $L_p$

Construct $M'$ from $(M, w)$
Rice's Theorem: Any nontrivial property of the RE languages is undecidable.

Proof: Let \( M_\emptyset \) be a TM that accepts the empty language. Then either \( \langle M_\emptyset \rangle \notin L_p \) or \( \langle M_\emptyset \rangle \in L_p \). In the former case, we prove that \( L_p \) is undecidable directly. In the latter case, we prove that \( L_p' \) is undecidable; this means that \( L_p \) is undecidable as well. So assume without loss of generality that \( \langle M_\emptyset \rangle \notin L_p \). Assume for the sake of contradiction that \( L_p \) is a decidable property of the RE languages.

Since \( L_p \) is a nontrivial property of the RE languages and \( \langle M_\emptyset \rangle \in L_p \), there must be some TM \( M_{yes} \) such that \( \langle M_{yes} \rangle \in L_p \). Moreover, \( L(M_{yes}) \neq L(M_\emptyset) \), because otherwise if \( L(M_{yes}) = L(M_\emptyset) \), we would have that either \( \langle M_\emptyset \rangle \in L_p \) and \( \langle M_{yes} \rangle \in L_p \) or \( \langle M_\emptyset \rangle \notin L_p \), both of which are false.

By Rice's Theorem: Any nontrivial property of the RE languages is undecidable.

Proof: …

Because \( L_p \) is decidable, let \( D \) be a decider for \( L_p \). Then consider the following TM:

\[ H = \text{"On input } \langle M, w \rangle:\]

1. Construct the TM \( M = \text{"On input } x:\)
   1. Run \( M \) on \( w \), and if \( M \) halts, run \( M_{yes} \) on \( x \).
   2. Accept if \( M_{yes} \) accepts, reject if \( M_{yes} \) rejects.
2. Run \( D \) on \( \langle M' \rangle \) and accept if \( D \) accepts and reject if \( D \) rejects.

We claim that \( H \) is a decider for \( HALT_{TM} \). If we can show this, then we have reached a contradiction because \( HALT_{TM} \) is undecidable. Thus our assumption was wrong, so no nontrivial property of the RE languages is decidable.

To see that \( H \) is a decider, note that since \( D \) is a decider, after we construct \( M' \) and run \( D \) on \( \langle M' \rangle \), \( D \) always halts, so \( H \) always halts. Thus \( H \) halts on all inputs.

To see that \( L(H) = HALT_{TM} \), note that if \( M \) loops on \( w \), then \( L(M') = \emptyset \), and so \( D \) rejects \( \langle M' \rangle \) because \( L(M') = \emptyset = L(M_\emptyset) \) and \( \langle M_\emptyset \rangle \notin L_p \). Otherwise, if \( M \) halts on \( w \), then \( M' \) accepts \( x \) iff \( M_{yes} \) accepts \( x \), so \( L(M') = L(M_{yes}) \), and since \( \langle M_{yes} \rangle \in L_p \), \( D \) accepts \( \langle M' \rangle \). Thus \( H \) accepts \( \langle M, w \rangle \) iff \( D \) accepts \( \langle M' \rangle \) iff \( M \) halts on \( w \) iff \( \langle M, w \rangle \in HALT_{TM} \), so \( L(H) = HALT_{TM} \).
Summary

Reductions from known undecidable problems like $A_{TM}$ and $HALT_{TM}$ can be used to prove that certain languages are undecidable.

Rice’s Theorem can be used to prove that large classes of languages are also undecidable.

Acknowledgments

This lecture incorporates material from:
Keith Schwarz