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:

- \textbf{Does not reject}:
  - Accept
  - \{ Halts (always) \}

- \textbf{Does not accept}:
  - Reject

\begin{center}
\begin{tabular}{c c}
\hline
Accept & \hline
\end{tabular}
\begin{tabular}{c c c}
\hline
Reject & \hline
\end{tabular}
\end{center}

\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 \textbf{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 \( \textbf{R} \) 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 \( \textbf{R} \) than in \( \textbf{RE} \).

\( \textbf{A}_{\text{TM}} \) and \( \text{HALT} \)

Both \( \textbf{A}_{\text{TM}} \) and \( \text{HALT} \) are undecidable.

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

However, both \( \textbf{A}_{\text{TM}} \) and \( \text{HALT} \) are recognizable.

- We can always run a TM on a string \( w \) and accept if that TM accepts or halts.

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

\( \textbf{RE} \) and co-\( \textbf{RE} \)

A language \( L \) is in \( \textbf{RE} \) iff there is a TM \( M \) such that

\begin{align*}
\text{if } w \in L, & \text{ then } M \text{ accepts } w \\
\text{if } w \notin L, & \text{ then } M \text{ does not accept } w
\end{align*}

A TM \( M \) of this sort is called a \textit{recognizer}, and \( L \) is called \textit{Turing-recognizable}.

A language \( L \) is in \( \text{co-RE} \) iff there is a TM such that

\begin{align*}
\text{if } w \in L, & \text{ then } M \text{ does not reject } w \\
\text{if } w \notin L, & \text{ then } M \text{ rejects } w
\end{align*}

A TM \( M \) of this sort is called a \textit{co-recognizer} and the language \( L \) is called \textit{co-Turing-recognizable}.
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*.
Reductions and decidability

Suppose we want to prove that some problem \( \text{harder} \) is undecidable.

If we can reduce an undecidable problem \( \text{hard} \) to \( \text{harder} \), then we know that we cannot decide \( \text{harder} \).

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 \( \text{HALT}_{TM} \in \text{RE} \) (i.e., is Turing-recognizable), but is not Turing-decidable:

\[
\text{HALT}_{TM} = \{ \langle M, w \rangle \mid M \text{ halts on } w \}
\]

If we can reduce \( \text{HALT}_{TM} \) 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 \( TOTAL = \{\langle M \rangle | M \text{ halts on all inputs} \}. \)

This seems at least as hard as the Halting Problem.

How could we prove that it's undecidable?

**Proof idea**: Reduce \( HALT_{TM} \) to \( TOTAL \).

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

Conclude that no such decider can exist.

---

**Intuition**

Machine \( M \)

Input \( w \)

**Build a new TM out of some old TM and a string**

**This new TM is total iff \( M \) accepts \( w \)**

---

**The Totality Problem**

Assume, for the sake of contradiction, that \( TOTAL \) is decidable.
We're going to try to show how a decider for TOTAL decides \( \text{HALT}_{TM} \), so our input must be a TM and a string.

\[ \langle M, w \rangle \]

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

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

\( x \)

(ignored)

\( M' = \) “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 \( \langle M, w \rangle \)

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

(ignored)

Decider for TOTAL

\( x \)

M' always halts

M halts on \( w \):

M' always halts

M loops on \( w \):

M' never halts

The Totality Problem

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

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

(ignored)

Decider for TOTAL

\( x \)

M halts on \( w \):

Always halts

M loops on \( w \):

(Always halts)

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

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

(Always halts)
The Totality Problem

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

$M'$

$\langle M' \rangle$

Decider for TOTAL

What does $D$ do if $M$ halts on $w$?

(Always halts)

The Totality Problem

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

$M'$

$\langle M' \rangle$

Decider for TOTAL

What does $D$ do if $M$ loops on $w$?

(Never halts)

The Totality Problem

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

$M'$

$\langle M' \rangle$

Decider for TOTAL

What does $D$ do if $M$ loops on $w$?

(Never halts)

The Totality Problem

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

$M'$

$\langle M' \rangle$

Decider for TOTAL

$D$ is a decider for $\text{HALT}_\text{TM}$!
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, \( \text{HALT}^\text{Tm} \)).

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:

\[ D = \text{“On input } \langle M, w \rangle \text{:} \]

1. Construct the TM \( M' = \text{‘On input } x \text{’} \):
   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 \( w \).
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 ⟨M, w⟩:
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.’
2. Run T on ⟨M’⟩.
3. If T accepts, accept.
4. If T rejects, reject.”

We claim that D decides HALT_{TM}. To see this, we show that D is a decider and that L(D) = HALT_{TM}. To see that D is a decider, note that after we construct M’, we run T on ⟨M’⟩. Since T is a decider, it always halts, so D always halts.

To see that L(D) = HALT_{TM}, note that D accepts ⟨M, w⟩ iff T accepts ⟨M’⟩. Because T is a decider for TOTAL, T accepts ⟨M’⟩ iff M’ halts on all inputs. By construction, M’ halts on any input iff M halts on w. Finally, M halts on w iff ⟨M, w⟩ ∈ HALT_{TM}. This means that D accepts ⟨M, w⟩ iff ⟨M, w⟩ ∈ HALT_{TM}, so L(D) = HALT_{TM}.

We have reached a contradiction, because D decides HALT_{TM}, which we know is undecidable. Thus our assumption was wrong and TOTAL is undecidable. ■

Example reduction:
REGULAR_{TM}

Testing regularity

We know that every 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

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

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

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

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

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

**Proof idea:** Suppose \( \text{REGULAR}_{TM} \) 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 \( \text{A}_{TM} \), which is a contradiction.

How do we build a machine with these properties?
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 $M_2$

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 $L(M_2) = \{0^n1^n | n \in \mathbb{N}_0\}$.

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 $M_2$

$M_2$

Building $M_2$

$M_2$

Building $M_2$

$M_2$

Building $M_2$

$M_2$

$M_2 = \text{"On input } x:"
1. If $x$ has the form $0^n1^n$, 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 $0^n1^n$. 
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 \( \theta^*1^n \), 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 \( L(S) = A_{TM} \).

\[
\text{We need to show both of these:}
\]

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 \( \theta^*1^n \), 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...
2. To see that \( L(S) = A_{TM} \), note that \( S \) 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 \( \theta^*1^n \) or because it accepts in the final step after \( M \) accepts \( w \). Thus \( L(M_2) = \Sigma^* \), which is regular.
   If \( M \) does not accept \( w \), then \( M \) only accepts \( x \) if it has the form \( \theta^*1^n \).
   Thus \( L(M_2) = \{ \theta^*1^n \mid n \in \mathbb{N}_0 \} \), which is not regular. Thus \( S \) accepts \( \langle M, w \rangle \) iff \( M \) accepts \( w \) iff \( \langle M, w \rangle \in A_{TM} \), so \( L(S) = A_{TM} \).

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

1. Construct the machine \( M_2 = \text{"On input } x\) :
   1. If \( x \) has the form \( \theta^*1^n \), 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...
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.
Rice's Theorem

Consider the following problems:

- **$L_D$:** Does $M$ reject $\langle M \rangle$?  
  Undecidable
- **$A_{TM}$:** Does $M$ accept $w$?  
  Undecidable
- **$HALT_{TM}$:** Does $M$ halt on $w$?  
  Undecidable
- **$TOTAL$:** Does $M$ halt on all inputs?  
  Undecidable
- **$REGULAR_{TM}$:** Is $L(M)$ regular?  
  Undecidable

There seems to be a trend here.

What properties of Turing machines are decidable?

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

For example:

- Does $L = \emptyset$?
- 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 \( \langle M_2 \rangle \notin L_P \), e.g.,

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

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

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

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

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 \( \langle M_2 \rangle \notin L_{\text{evenQ}} \)

**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:
  $\exists M_1 . \exists M_2 . \langle M_1 \rangle \in L_{ne} \land \langle M_2 \rangle \notin L_{ne}$

✓ $L_{ne}$ 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?

$L_{es} = \{\langle M \rangle \mid M$ has an even number of states$\}$

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

✓ $L_{es}$ is nontrivial:
  $\exists M_1 . \exists M_2 . \langle M_1 \rangle \in L_{es} \land \langle M_2 \rangle \notin L_{es}$

✓ $L_{es}$ 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:
  $\exists M_1 . \exists M_2 . \langle M_1 \rangle \in L_{es} \land \langle M_2 \rangle \notin L_{es}$

✓ $L_{small}$ 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:

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

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

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

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

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

$E_{TM} = \{\langle M \rangle \mid L(M) = \emptyset\}$
Acknowledgments

This lecture incorporates material from:

Keith Schwarz