An engineer and a mathematician are out for a walk when they spot a house on fire. There is a hose lying nearby; the engineer quickly attaches it to a fire hydrant and puts out the fire. They continue walking. The mathematician spots a house where there is a hose attached to a hydrant. He detaches the hose and sets the house on fire, thus reducing it to a previously solved problem.
Recall: TMs and Languages

✓ A TM **accepts** a string if that input leads to the “y” state.

✓ A language is **semidecidable** (a.k.a. semidecidable, recursively enumerable) if there is a TM that accepts exactly the strings in the language.

✓ A language is **decidable** (a.k.a. recursive) if it is accepted by a TM that always halts (i.e., the TM always ends up in y or n).
Recall: Decidable vs. Semidecidable

✓ If a language is decidable, then its complement is decidable. Why?

✓ If a language is semidecidable, and its complement is semidecidable, then the language is decidable. Why?
**Turing Machines as Enumerators**

Several variant definitions. Each specify a language \( L \).

1. A TM that prints out all the members of \( L \), one at a time (but not necessarily in any particular order)

2. A TM that prints out all the members of \( L \), one at a time (but...) with arbitrarily many repeats.

3. A TM that, given an integer \( n \), returns the \( n \)th element of a sequence like (1) above.

4. A TM that, given an integer \( n \), returns the \( n \)th element of a sequence like (2) above.

For a fixed language, all these are interconvertable.

**Theorem**

A language is semidecidable \( \iff \) it can be enumerated.

A language is decidable \( \iff \) it can be enumerated in lexicographic order.
Church-Turing Thesis

If it can be done at all, then it can be done by

✓ A Turing Machine
✓ Lambda Calculus
✓ An Unrestricted Grammar
✓ A 2-register machine
✓ C
✓ …

(Note: assumes suitably coded inputs and outputs)
Is There More?

Regular
\[ a^*b^* \]

Context Free
\[ a^n b^n \]

Decidable
\[ a^p b^p c^p \]
(p perfect)
LANGUAGES OF ACCEPTANCE

Which are semidecidable (by a TM)? Decidable?

✓ $A_{DFA} = \{ \langle D, w \rangle \mid D \text{ a DFA}, D \text{ accepts } w \}$
✓ $A_{NFA} = \{ \langle N, w \rangle \mid N \text{ an NFA}, N \text{ accepts } w \}$
✓ $A_{RE} = \{ \langle R, w \rangle \mid R \text{ a regexp}, R \text{ matches } w \}$
✓ $A_{CFG} = \{ \langle G, w \rangle \mid G \text{ a CFG}, G \text{ produces } w \}$
✓ $A_{TM} = \{ \langle M, w \rangle \mid M \text{ a TM}, M \text{ accepts } w \}$
Digression: Bootstrapping a Compiler

Lots of compilers are written in the same language they compile!

✓ Gnu C Compiler (used in CS 105) is written in C
✓ Glasgow Haskell Compiler (used in CS 131) is in Haskell
✓ etc.

Practical reasons to run programs on their own source code!
**ATM is not decidable**

<table>
<thead>
<tr>
<th></th>
<th>$\langle M_0 \rangle$</th>
<th>$\langle M_1 \rangle$</th>
<th>$\langle M_2 \rangle$</th>
<th>$\langle M_3 \rangle$</th>
<th>$\langle M_4 \rangle$</th>
<th>$\cdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_0$</td>
<td>Acc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_1$</td>
<td></td>
<td>Acc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_2$</td>
<td>Acc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M_4$</td>
<td></td>
<td>Acc.</td>
<td></td>
<td>Acc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Is There More?

- Regular: $a^*b^*$
- Context Free: $a^n b^n$
- Decidable: $a^p b^p c^p$ (p perfect)
- Semidecidable: $A_{TM}$
What is the complement of $A_{TM}$?
We’ll Stop Here
Obligatory Corollary

Theorem
The language
\[ H = \{ \langle M, w \rangle \mid M \text{ a TM, } M \text{ halts on } w \} \]
is not decidable.

Proof.
Suppose there were a halt-checking TM...
Undecidability, So Far

1. Acceptance for TMs is semidecidable but not decidable.

\[ A_{TM} = \{ \langle M, w \rangle \mid M \text{ a TM, } M \text{ accepts } w \} \]

2. Non-Acceptance for TMs is not semidecidable (hence not decidable).

3. TM Halting is semidecidable but not decidable.

\[ H = \{ \langle M, w \rangle \mid M \text{ a TM, } M \text{ halts on } w \} \]
**Reductions**

Given problems $P$ and $Q$, we say that

$$P \leq Q$$

if a solution to $Q$ would let us solve $P$ as well.

✓ I.e., (P is “not fundamentally more difficult” Q.)

✓ We say “$P$ reduces to $Q$.”

In Math, we often show we can solve a problem $X$ by taking advantage of previously-solved problem $Q$ (i.e., prove $X \leq Q$)

In Theocomp, we typically prove a problem $X$ hard by showing a solution to $X$ would also solve the hard problem $P$ (i.e., prove $P \leq X$).
**Reductions**

To prove that $P$ reduces to $Q$ ($P \leq Q$), it suffices to prove:

- If we have a solver for $Q$, then we can use it to solve any instance of $P$.
- I.e., show you could construct a $P$-solver if you could make calls to a $Q$-solving subroutine.

Commonly, we instead prove a “mapping reduction”:

- For every instance of $P$, we can construct an instance of $Q$ with the same yes/no answer.

Why is this enough?
**Warning**

It is easy to get the reduction backwards!

These are essentially proof by contradiction:

✓ Assume the unknown problem $X$ is decidable/semidecidable
✓ Show that it means that $H$ (or $A_{TM}$ or ...) is decidable/semidecidable.
✓ Contradiction.
✓ Therefore, $X$ is not decidable/semidecidable.

If you ever find yourself assuming things known to be false ("assume I had a halt checker... then I could solve $X$") you’re doing the reduction wrong.
**Reduction Practice**

Show the following are not decidable (e.g., by reducing $A_{TM}$ to each).

- $NE_{TM} := \{ \langle M \rangle \mid M \text{ accepts at least one } w \in \Sigma^* \}$
- $E_{TM} := \{ \langle M \rangle \mid L(M) = \emptyset \}$
- $ALL_{TM} := \{ \langle M \rangle \mid L(M) = \Sigma^* \}$
- $Accepts-s := \{ \langle M \rangle \mid M \text{ accepts } s \}$
- $Regular := \{ \langle M \rangle \mid L(M) \text{ is regular} \}$
**FUNCTIONAL vs. STRUCTURAL PROPERTIES**

Structural questions (often, but not always, decidable):

- ✓ Does this TM have more than 500 control states?
- ✓ Does this TM ever use more than 300 cells of tape for some input?
- ✓ Does this TM ever write the symbol $a$?

Functional questions (rarely decidable):

- ✓ Does this TM accept the empty language?
- ✓ Does this TM accept a finite language?
- ✓ Do all the strings accepted by this TM have even length?
- ✓ In general:

  Given a TM $T$, does $L(T)$ have property $P$?
  Given a semidecidable $L$, does $L$ have property $P$?
Rice’s Theorem

No functional and nontrivial property of Turing Machines is decidable.

That is, no nontrivial property of recognizable languages is decidable.
Consequences?

“That is, no nontrivial property of recognizable languages is decidable.”

✓ $\text{NE}_{TM} := \{ \langle M \rangle \mid M \text{ accepts at least one } w \in \Sigma^* \}$
✓ $\text{E}_{TM} := \{ \langle M \rangle \mid L(M) = \emptyset \}$
✓ $\text{ALL}_{TM} := \{ \langle M \rangle \mid L(M) = \Sigma^* \}$
✓ $\text{Accepts-s} := \{ \langle M \rangle \mid M \text{ accepts } s \}$
✓ $\text{Regular} := \{ \langle M \rangle \mid L(M) \text{ is regular} \}$
Proof (Sketch) of Rice’s Theorem

Setup: we have a nontrivial property \( P(L) \) of languages.

✓ WLOG, \( \neg P(\emptyset) \).
✓ But some TM’s language satisfies \( P \). Pick a TM \( K \) such that \( P(L(K)) \).

Plan: find a mapping reduction from \( A_{\text{TM}} \) to \( P \).

Given \( \langle M, w \rangle \), construct \( M' \), which does the following:

1. Take input \( x \). Save it an an auxiliary tape.
2. Run \( M \) on \( w \).
3. If \( M \) accepts \( w \), run \( K \) on \( x \). Otherwise, reject.

Claim: \( M \) accepts \( w \) if and only if \( L(M') \) has property \( P \).
That is, if we had a \( P \)-checker, we could decide \( A_{\text{TM}} \).
Limitations of Rice’s Theorem

✓ It must be a functional property (i.e., expressible as a property about languages)

✓ It must be a nontrivial property among the languages accepted by TMs

✓ If so, the property is not decidable. But Rice’s Theorem tells us nothing about whether the property is semidecidable or not.
CS 70 Grader Permanent Employment Theorem

The language

\[ \text{EQ}_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid L(M_1) = L(M_2) \} \]

is not decidable.

The same holds for pairs of programs written in C++, Java, etc.
Theorem

There is no perfect size-optimizing compiler.

Proof.

Any program that infinite loops without output could be identified, as it would reduce to a single loop instruction:

L1: jmp L1
Perfect Garbage Collection is Undecidable

Java, Python, Haskell, Scheme, etc., all rely on garbage collection to deallocate unused memory.

✓ At any point during execution, a piece of data is live if it will be used in the future, and otherwise dead or garbage.

✓ A garbage collector detects and deallocates garbage.

✓ Perfect garbage collection is undecidable.