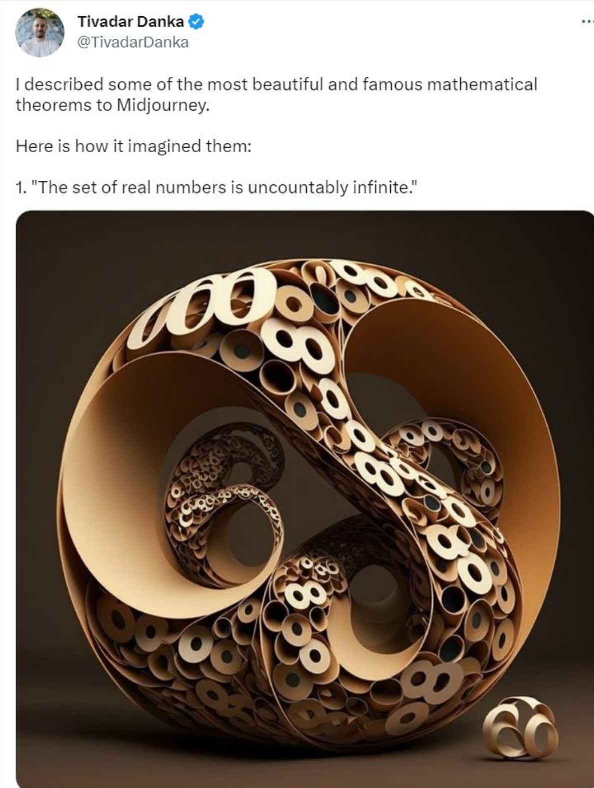
# CS420

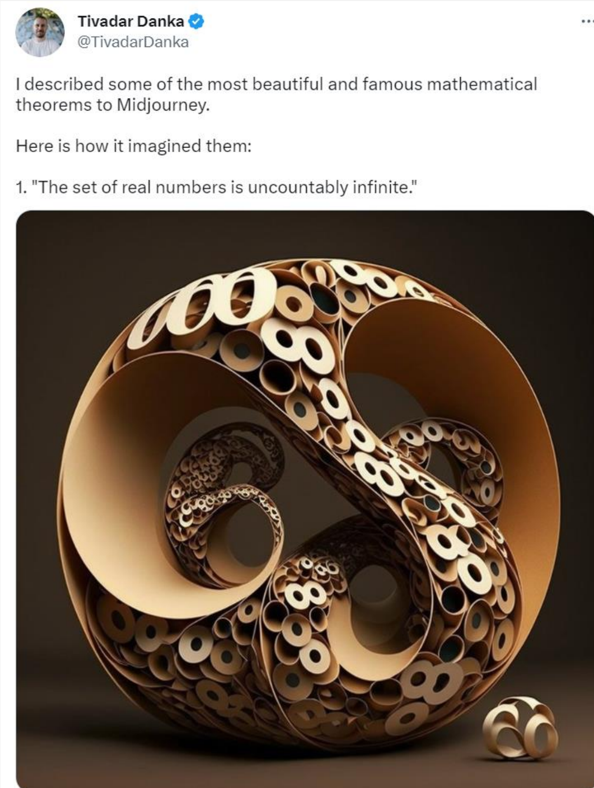
# Reducibility

Wednesday, April 24, 2024



# Announcements

- HW 9 in
  - ~~Due Wed 4/24 12pm noon~~
- HW 10 out
  - Due Wed 5/1 12pm noon



Last Time

3 Easy Steps!

# Thm: $A_{\mathsf{TM}}$ is undecidable

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

Proof by contradiction:

1. **Assume $A_{\mathsf{TM}}$ is decidable.** So there exists a decider $H$ for it:

$$H(\langle M, w\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

Using Examples (Tables) to understand these kinds of problems are critical!

2. **Use $H$ in another TM** … the **impossible "opposite" machine:**

$D$ = "On input $\langle M\rangle$, where $M$ is a TM:
1. Run $H$ on input $\langle M, \langle M\rangle\rangle$.
2. Output the opposite of what $H$ outputs. That is, if $H$ accepts, *reject*; and if $H$ rejects, *accept*."

$H$ computes: $M$'s result with $\langle M\rangle$ as input

$D$ returns opposite of $H$

$D$ result with input $\langle D\rangle$?
- If $D$ accepts $\langle D\rangle$, then $D$ rejects $\langle D\rangle$
- If $D$ rejects $\langle D\rangle$, then $D$ accepts $\langle D\rangle$

Last Time

3 Easy Steps!

# Thm: $A_{\mathsf{TM}}$ is undecidable

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

Proof by contradiction:

This cannot be true

1. **Assume** $A_{\mathsf{TM}}$ **is decidable.** So **there exists a decider** $H$ **for it:**

$$H(\langle M, w\rangle) = \begin{cases} \mathit{accept} & \text{if } M \text{ accepts } w \\ \mathit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

2. **Use** $H$ **in another TM** … the **impossible "opposite" machine:**

$D =$ "On input $\langle M\rangle$, where $M$ is a TM:
   1. Run $H$ on input $\langle M, \langle M\rangle\rangle$.
   2. Output the opposite of what $H$ outputs. That is, if $H$ accepts, *reject*; and if $H$ rejects, *accept*."

3. So $D$ does not exist! **Contradiction**! So the assumption is false.

# Easier Undecidability Proofs

- **We proved** $A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ **undecidable …**
- … by **contradiction:**
  - **Use** hypothetical $A_{\mathsf{TM}}$ **decider to** create an impossible decider “$D$”!

**reduce “$D$ problem” to $A_{\mathsf{TM}}$**

- Step # 1: **coming up with “$D$”** --- hard!
  - Need to invent **diagonalization**

| | $\langle M_1\rangle$ | $\langle M_2\rangle$ | $\langle M_3\rangle$ | $\langle M_4\rangle$ | $\cdots$ | $\langle D\rangle$ |
|---|---|---|---|---|---|---|
| $M_1$ | *accept* | *reject* | *accept* | *reject* | | *accept* |
| $M_2$ | *accept* | *accept* | *accept* | *accept* | $\cdots$ | *accept* |
| $M_3$ | *reject* | *reject* | *reject* | *reject* | | *reject* |
| $M_4$ | *accept* | *accept* | *reject* | *reject* | | *accept* |
| $\vdots$ | | $\vdots$ | | | $\ddots$ | |
| $D$ | *reject* | *reject* | *accept* | *accept* | | ? |

Known undecidable lang!

- Step # 2: **reduce “$D$” problem to** $\cancel{A_{\mathsf{TM}}}$ --- easier!

- From now on: **undecidability proofs only need** step # 2**!**
  - And we now **have two “impossible” problems to choose from**

# The Halting Problem

$$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Thm: $HALT_{\mathsf{TM}}$ is undecidable

Proof, by **contradiction**:

- Assume: $HALT_{\mathsf{TM}}$ has *decider* $R$; use it to create decider for $A_{\mathsf{TM}}$:

Examples Table(s) are critical for these kinds of problems!

Let $\langle M, w\rangle$ be a string where:
- $M$ is some TM and
- $w$ is some string

Example Table for $R$

| String | $M$ on $w$ | $R$ on $\langle M, w\rangle$ | In lang $HALT_{\mathsf{TM}}$? |
|---|---|---|---|
| $\langle M, w\rangle$ | (halt and) Accept | Accept | Yes |
| $\langle M, w\rangle$ | (halt and) Reject | Accept | Yes |
| $\langle M, w\rangle$ | Loop | Reject | No |

# The Halting Problem

$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

Thm: $HALT_{\mathsf{TM}}$ is undecidable

Proof, by **contradiction**:

**reduce** (from known **undecidable**) $A_{\mathsf{TM}}$ to $HALT_{\mathsf{TM}}$

- Assume: $HALT_{\mathsf{TM}}$ has *decider* $R$; use it to create decider for $A_{\mathsf{TM}}$:

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

- ...

contradiction

- But $A_{TM}$ is undecidable and has no decider!

# The Halting Problem

$HALT_{\mathsf{TM}} = \{\langle M, w\rangle |\ M \text{ is a TM and } M \text{ halts on input } w\}$

Thm: $HALT_{TM}$ is undecidable

Proof, by **contradiction**:

Using our hypothetical $HALT_{TM}$ decider $R$

- Assume: $HALT_{TM}$ has *decider* $R$; use it to create decider for $A_{TM}$:

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

$S$ = "On input $\langle M, w\rangle$, an encoding of a TM $M$ and a string $w$:

**1.** Run TM $R$ on input $\langle M, w\rangle$.

**2.** If $R$ rejects, *reject*.

If $R$ rejects ⟨$M$,$w$⟩, $M$ loops on input $w$, so $S$ rejects

**3.** If $R$ accepts, simulate $M$ on $w$ until it halts.

This step always halts

**4.** If $M$ has accepted, *accept*; if $M$ has rejected, *reject*."

Examples Table??

Termination argument:
**Step 1**: $R$ is a decider so always halts
**Step 3**: $M$ always halts because $R$ said so

# The Halting Problem

These must match (like before)

$HALT_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

Let $\langle M, w\rangle$ be a string where:
- $M$ is some TM and
- $w$ is some string

| String | $M$ on $w$ | $HALT_{\mathrm{TM}}$ decider $R$ on $\langle M, w\rangle$ | $A_{\mathrm{TM}}$ decider $S$ on $\langle M, w\rangle$ | In lang $A_{\mathrm{TM}}$? |
|---|---|---|---|---|
| $\langle M, w\rangle$ | Accept | Accept | Accept | Yes |
| $\langle M, w\rangle$ | Reject | Accept | Reject | No |
| $\langle M, w\rangle$ | Loop | Reject | Reject | No |

Example Table for $A_{\mathrm{TM}}$ decider $S$

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

$S$ = "On input $\langle M, w\rangle$, an encoding of a TM $M$ and a string $w$:
1. Run TM $R$ on input $\langle M, w\rangle$.
2. If $R$ rejects, *reject*.
3. If $R$ accepts, simulate $M$ on $w$ until it halts.
4. If $M$ has accepted, *accept*; if $M$ has rejected, *reject*."

Now these must (sometimes) match

Because we are using $R$ ($HALT_{\mathrm{TM}}$) to help decide $A_{\mathrm{TM}}$!!

Examples Table??

Undecidability Proof Technique #1:
**Reduce** from **known undecidable language** (by **creating its decider**)

# The Halting Problem

$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

Thm: $HALT_{\mathsf{TM}}$ **is undecidable**

Proof, by **contradiction**:

- Assume: $HALT_{\mathsf{TM}}$ **has** *decider R*; **use it to create decider for** $A_{\mathsf{TM}}$:

$S$ = "On input $\langle M, w\rangle$, an encoding of a TM $M$ and a string $w$:
1. Run TM $R$ on input $\langle M, w\rangle$.
2. If $R$ rejects, *reject*.
3. If $R$ accepts, simulate $M$ on $w$ until it halts.
4. If $M$ has accepted, *accept*; if $M$ has rejected, *reject*."

Now we have <u>three</u> **known undecidable langs**, i.e., **three "impossible" deciders**, to choose from

- But $A_{TM}$ **is undecidable** (has no decider)! I.e., **this decider does not exist!**
  - **So** $HALT_{\mathsf{TM}}$ **is also undecidable!**

# The Halting Problem … As Statements / Justifications

$HALT_{\mathsf{TM}} = \{\langle M, w\rangle |\ M \text{ is a TM and } M \text{ halts on input } w\}$

(Proof by contradiction)

| **Statements** | **Justifications** |
|---|---|
| 1. $HALT_{\mathsf{TM}}$ is decidable | 1. Opposite of statement to prove |
| 2. $HALT_{\mathsf{TM}}$ has decider $R$ | 2. Definition of decidable langs |
| 3. Construct decider $S$ using $R$ ("see below") | 3. Definition of TMs and deciders (incl termination argument) |
| 4. Decider $S$ decides $A_{\mathsf{TM}}$ | 4. See Examples Table |
| 5. $A_{\mathsf{TM}}$ is undecidable (i.e, it has no decider) | 5. Theorem from last lecture (Sipser Theorem 4.11) |
| 6. $HALT_{\mathsf{TM}}$ is undecidable | 6. Contradiction of Stmts #4 & #5 |

# *Interlude:* Reducing from $HALT_{TM}$

A practical thought experiment …
… about **compiler optimizations**

Your compiler changes your program!

```
If TRUE then A else B   ⇨   A

            1 + 2 + 3   ⇨   6
```

# Compiler Optimizations

**Optmization** - docs

- -O0
  - No optmization, faster compilation time, better for debugging builds.
- -O2
- -O3
  - Higher level of optmization. Slower compile-time, better for production builds.
- -OFast
  - Enables higher level of optmization than (-O3). It enables lots of flags as can be seen src (-ffloat-store, -ffsast-math, -ffinite-math-only, -O3 …)
- -finline-functions
- -m64
- -funroll-loops
- -fvectorize
- -fprofile-generate

## Types of optimization [ edit ]

Techniques used in optimization can be broken up among various *scopes* which can affect anything from a single statement to the entire program. Generally speaking, locally scoped techniques are easier to implement than global ones but result in smaller gains. Some examples of scopes include:

**Peephole optimizations**

These are usually performed late in the compilation process after machine code has been generated. This form of optimization examines a few adjacent instructions (like "looking through a peephole" at the code) to see whether they can be replaced by a single instruction or a shorter sequence of instructions.[2] For instance, a multiplication of a value by 2 might be more efficiently executed by left-shifting the value or by adding the value to itself (this example is also an instance of strength reduction).

**Local optimizations**

These only consider information local to a basic block.[3] Since basic blocks have no control flow, these optimizations need very little analysis, saving time and reducing storage requirements, but this also means that no information is preserved across jumps.

**Global optimizations**

These are also called "intraprocedural methods" and act on whole functions.[3] This gives them more information to work with, but often makes expensive computations necessary. Worst case assumptions have to be made when function calls occur or global variables are accessed because little information about them is available.

**Loop optimizations**

These act on the statements which make up a loop, such as a *for* loop, for example loop-invariant code motion. Loop optimizations can have a significant impact because many programs spend a large percentage of their time inside loops.[4]

**Prescient store optimizations**

These allow store operations to occur earlier than would otherwise be permitted in the context of threads and locks. The process needs some way of knowing ahead of time what value will be stored by the assignment that it should have followed. The purpose of this relaxation is to allow compiler optimization to perform certain kinds of code rearrangement that preserve the semantics of properly synchronized programs.[5]

**Interprocedural, whole-program or link-time optimization**

These analyze all of a program's source code. The greater quantity of information extracted means that optimizations can be more effective compared to when they only have access to local information, i.e. within a single function. This kind of optimization can also allow new techniques to be performed. For instance, function inlining, where a call to a function is replaced by a copy of the function body.

**Machine code optimization and object code optimizer**

These analyze the executable task image of the program after all of an executable machine code has been linked. Some of the techniques that can be applied in a more limited scope, such as macro compression which saves space by collapsing common sequences of instructions, are more effective when the entire executable task image is available for analysis.[6]

# The Optimal Optimizing Compiler

"Full Employment" Theorem

Thm: **The Optimal (C++) Optimizing Compiler does not exist**

Proof, by **contradiction**:

Assume: $OPT$ **is the Perfect Optimizing Compiler**

Use it **to create** $HALT_{TM}$ **decider** (**accept**s $<M,w>$ if $M$ halts with $w$, else **reject**s):

$S$ = **On input** $<M, w>$, where $M$ is **C++** program and $w$ is string:

- If $OPT(M)$ == `for(;;)`
  a) Then **Reject**
  b) Else **Accept**

In computer science and mathematics, a **full employment theorem** is a term used, often humorously, to refer to a theorem which states that no algorithm can optimally perform a particular task done by some class of professionals. The name arises because such a theorem ensures that there is endless scope to keep discovering new techniques to improve the way at least some specific task is done.

For example, the *full employment theorem for compiler writers* states that there is no such thing as a provably perfect size-optimizing compiler, as such a proof for the compiler would have to detect non-terminating computations and reduce them to a one-instruction infinite loop. Thus, the existence of a provably perfect size-optimizing compiler would imply a solution to the halting problem, which cannot exist. This also implies that there may always be a better compiler since the proof that one has the best compiler cannot exist. Therefore, compiler writers will always be able to speculate that they have something to improve.

# *Summary:* The Limits of Algorithms

- $A_{\mathsf{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$ Decidable
- $A_{\mathsf{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ Decidable
- $A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ **Undecidable**
- $HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$ **Undecidable**

Similar languages

It's straightforward to use hypothetical $HALT_{\mathsf{TM}}$ decider to create $A_{\mathsf{TM}}$ decider

# *Summary:* The Limits of Algorithms

- $A_{\mathsf{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$ Decidable
- $A_{\mathsf{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ Decidable
- $A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ **Undecidable**
- $HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$ **Undecidable**
- $E_{\mathsf{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$ Decidable
- $E_{\mathsf{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$ Decidable
- next $E_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$ **Undecidable**

Not as similar languages

How can we use a hypothetical $E_{\mathsf{TM}}$ decider to create $A_{\mathsf{TM}}$ or $HALT_{\mathsf{TM}}$ decider?

Examples Table(s) are critical here!

Undecidability Proof Technique #2

# Reducibility: Modifying the TM

$E_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

Thm: $E_{\mathsf{TM}}$ is undecidable

Proof, by **contradiction**:

- Assume $E_{\mathsf{TM}}$ has *decider* $R$; use it to create *decider* for $A_{\mathsf{TM}}$:

$S =$ "On input $\langle M, w\rangle$, an encoding of a TM $M$ and a string $w$:

- Run $R$ on input $\langle M\rangle$
- If $R$ accepts, *reject* (because it means $\langle M\rangle$ doesn't accept anything)
- if $R$ rejects, then **???** ($\langle M\rangle$ accepts something, but is it $w$???)

Now **these must match** (sometimes), **but …?** ☒

**These must match** (like before) ☑

Let $\langle M, w\rangle$ be a string where:
- $M$ is some TM and
- $w$ is some string

| String | $M$ on $w$ | $R$ on $\langle M\rangle$ | $S$ on $\langle M, w\rangle$ | In lang $A_{\mathsf{TM}}$? |
|---|---|---|---|---|
| $\langle M, w\rangle$ | Accept | ?? | Accept | Yes |
| $\langle M, w\rangle$ | Reject | ?? | Reject | No |
| $\langle M, w\rangle$ | Loop | ?? | Reject | No |

Example Table for $A_{\mathsf{TM}}$ decider $S$

# Reducibility: Modifying the TM

$E_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

Thm: $E_{TM}$ is undecidable

Proof, by contradiction:

- Assume $E_{TM}$ has *decider* $R$; use it to create *decider* for $A_{TM}$:

  $S =$ "On input $\langle M, w \rangle$, an encoding of a TM $M$ and a string $w$:

  - Run $R$ on input $\langle M \rangle$
  - If $R$ accepts, *reject* (because it means $\langle M \rangle$ doesn't accept anything)
  - if $R$ rejects, then **???** ($\langle M \rangle$ accepts something, but is it $w$???)

- Idea: Wrap $\langle M \rangle$ in a new TM that can *only* (maybe) accept $w$:

$M_1 =$ "On input $x$:

1. If $x \neq w$, *reject*.
2. If $x = w$, run $M$ on input $w$ and *accept* if $M$ does."

Input not $w$, always reject

Input is $w$, maybe accept

$M_1$ accepts $w$ if $M$ does

$L(M_1)$ depends on $M$ and $w$!
If $M$ accepts $w$, $L(M_1) = \{w\}$
else $L(M_1) = \{\}$

Examples Table(s) are critical here!

Undecidability Proof Technique #2

# Reducibility: Modifying the TM

$E_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

Thm: $E_{\mathsf{TM}}$ is undecidable

Proof, by contradiction

Now opposites! ☑

- Assume $E_{\mathsf{TM}}$ has *decider* $R$; use it to create *decider* for $A_{\mathsf{TM}}$:

  $S =$ "On input $\langle M, w\rangle$, an encoding of a TM $M$ and a string $w$:

| String $x$ | $M$ on $w$ | $R$ on $\langle M_1\rangle$ | $M_1$ on $x$ | In lang $\{w\} \cap L(M)$ ? |
|---|---|---|---|---|
| $w$ | Accept | Reject | Accept | Yes (lang = $\{w\}$) |
| $w$ | Reject | Accept | Reject | No (lang = $\{\}$) |
| not $w$ | - | - | Reject | No |

Example Table for $M_1$

$L(M_1)$ depends on $M$ and $w$!
If $M$ accepts $w$, $L(M_1) = \{w\}$
else $L(M_1) = \{\}$

- Idea: Wrap $\langle M \rangle$ in a new TM that can *only* (maybe) accept $w$:

$M_1 =$ "On input $x$:
1. If $x \neq w$, *reject*.
2. If $x = w$, run $M$ on input $w$ and *accept* if $M$ does."

# Reducibility: Modifying the TM

$E_{\mathrm{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

Thm: $E_{\mathrm{TM}}$ is undecidable

Proof, by **contradiction**:

- Assume $E_{\mathrm{TM}}$ has *decider* $R$; use it to create *decider* for $A_{\mathrm{TM}}$:

$S$ = "On input $\langle M, w\rangle$, an encoding of a TM $M$ and a string $w$:

First, construct $M_1$

- Run $R$ on input $\langle M_1\rangle$

**Note**: $M_1$ is only used as arg to $R$; it's never run (avoiding loop)!

- If $R$ accepts, *reject* (because it means $\langle M\rangle$ doesn't accept $w$)
- if $R$ rejects, then *accept* ($\langle M\rangle$ accepts $w$

$L(M_1)$ depends on $M$ and $w$!
If $M$ accepts $w$, $L(M_1) = \{w\}$
else $L(M_1) = \{\}$

- Idea: Wrap $\langle M\rangle$ in a new TM that can *only* (maybe) accept $w$:

$M_1$ = "On input $x$:
1. If $x \neq w$, *reject*.
2. If $x = w$, run $M$ on input $w$ and *accept* if $M$ does."

# Reducibility: Modifying the TM

$E_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

Thm: $E_{\mathsf{TM}}$ is undecidable

Proof, by contradiction:

This decider for $A_{\mathsf{TM}}$ cannot exist!

- Assume $E_{\mathsf{TM}}$ has *decider* $R$; use it to create *decider* for $A_{\mathsf{TM}}$:

$S =$ "On input $\langle M, w\rangle$, an encoding of a TM $M$ and a string $w$:

First, construct $M_1$

- Run $R$ on input $\langle M_1\rangle$
- If $R$ accepts, *reject* (because it means $\langle M\rangle$ doesn't accept $w$)
- if $R$ rejects, then *accept* ($\langle M\rangle$ accepts $w$)

- Idea: Wrap $\langle M\rangle$ in a new TM that can *only* (maybe) accept $w$:

$M_1 =$ "On input $x$:

1. If $x \neq w$, *reject*.
2. If $x = w$, run $M$ on input $w$ and *accept* if $M$ does."

# *Summary:* The Limits of Algorithms

- $A_{\mathsf{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$ Decidable
- $A_{\mathsf{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ Decidable
- $A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ **Undecidable**
- $E_{\mathsf{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$ Decidable
- $E_{\mathsf{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$ Decidable
- $E_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$ needs **Undecidable**
- $EQ_{\mathsf{DFA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ Decidable
- $EQ_{\mathsf{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$ **Undecidable**

next
- $EQ_{\mathsf{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$ **Undecidable**

# Reduce to something else: $EQ_{\mathsf{TM}}$ is undecidable

$EQ_{\mathsf{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

Proof, by **contradiction**:

- Assume: $EQ_{\mathsf{TM}}$ has *decider* $R$; use it to create *decider* for ~~$A_{\mathsf{TM}}$~~ $E_{\mathsf{TM}}$:

$E_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

$S$ = "On input $\langle M\rangle$, where $M$ is a TM:

1. Run $R$ on input $\langle M, M_1\rangle$, where $M_1$ is a TM that rejects all inputs.
2. If $R$ accepts, *accept*; if $R$ rejects, *reject*."

# Reduce to something else: $EQ_{\mathsf{TM}}$ is undecidable

$EQ_{\mathsf{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

Proof, by **contradiction**:

- Assume: $EQ_{\mathsf{TM}}$ has *decider* $R$; use it to create *decider* for $E_{\mathsf{TM}}$:

$= \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

$S =$ "On input $\langle M\rangle$, where $M$ is a TM:

1. Run $R$ on input $\langle M, M_1\rangle$, where $M_1$ is a TM that rejects all inputs.
2. If $R$ accepts, *accept*; if $R$ rejects, *reject*."

- But $E_{\mathsf{TM}}$ is undecidable!

# *Summary:* Undecidability Proof Techniques

- Proof Technique #1:
  - Use hypothetical decider to implement impossible $A_{\mathsf{TM}}$ decider
  - Example Proof: $HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

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

Reduce

- Proof Technique #2:
  - Use hypothetical decider to implement impossible $A_{\mathsf{TM}}$ decider
  - But first modify the input $M$
  - Example Proof: $E_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

Reduce

Can also combine these techniques

- Proof Technique #3:
  - Use hypothetical decider to implement non-$A_{\mathsf{TM}}$ impossible decider
  - Example Proof: $EQ_{\mathsf{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

# *Summary:* Decidability and Undecidability

- $A_{\text{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$ Decidable
- $A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ Decidable
- $A_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ **Undecidable**
- $E_{\text{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$ Decidable
- $E_{\text{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$ Decidable
- $E_{\text{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$ **Undecidable**
- $EQ_{\text{DFA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ Decidable
- $EQ_{\text{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$ **Undecidable**
- $EQ_{\text{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$ **Undecidable**

# Also Undecidable …

next

- $REGULAR_{\mathrm{TM}}$ = {<$M$> | $M$ is a TM and $L(M)$ is a regular language}

Undecidability Proof Technique #2:
Modify input TM $M$

# Thm: $REGULAR_{\mathsf{TM}}$ is undecidable

$REGULAR_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language}\}$

Proof, by **contradiction**:

- Assume: $REGULAR_{\mathsf{TM}}$ has *decider* $R$; use it to create *decider* for $A_{\mathsf{TM}}$:

$S =$ "On input $\langle M, w\rangle$, an encoding of a TM $M$ and a string $w$:

  - First, construct $M_2$ (??)
  - Run $R$ on input $\langle M_2 \rangle$
  - If $R$ accepts, *accept*; if $R$ rejects, *reject*

Want: $L(M_2)$ =
- **regular**, if $M$ accepts $w$
- **nonregular**, if $M$ does not accept $w$

# <u>Thm</u>: $REGULAR_{\mathsf{TM}}$ is undecidable (continued)

$REGULAR_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language}\}$

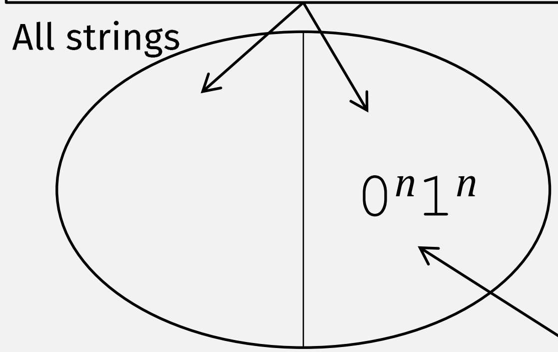
$M_2 =$ "On input $x$:

1. If $x$ has the form $0^n 1^n$, *accept*.
2. If $x$ does not have this form, run $M$ on input $w$ and *accept* if $M$ accepts $w$."

Always accept strings $0^n1^n$
$L(M_2)$ = **nonregular**, so far

If $M$ accepts $w$,
accept everything else,
so $L(M_2) = \Sigma^* =$ **regular**

if $M$ <u>does not</u> accept $w$, $M_2$ accepts all strings (**regular** lang)




if $M$ accepts $w$, $M_2$ accepts this **nonregular** lang

<u>Want</u>: $L(M_2)$ =

- **regular**, if $M$ accepts $w$
- **nonregular**, if $M$ does not accept $w$

# Also Undecidable …

Seems like no algorithm can compute **anything** about the language of a Turing Machine, i.e., about the runtime behavior of programs …

- $REGULAR_{\mathrm{TM}}$ = {<$M$> | $M$ is a TM and $L(M)$ is a regular language}
- $CONTEXTFREE_{\mathrm{TM}}$ = {<$M$> | $M$ is a TM and $L(M)$ is a CFL}
- $DECIDABLE_{\mathrm{TM}}$ = {<$M$> | $M$ is a TM and $L(M)$ is a decidable language}
- $FINITE_{\mathrm{TM}}$ = {<$M$> | $M$ is a TM and $L(M)$ is a finite language}

# An Algorithm About Program Behavior?

```
main()
{
    printf("hello, world\n");
}
```

**Write a program** that,
given another program as its argument,
returns TRUE if that argument prints
“`Hello, World!`”

TRUE

**Fermat's Last Theorem**
(unknown for ~350 years,
solved in 1990s)

main()
{
  If $\boxed{x^n + y^n = z^n, \text{ for any integer } n > 2}$
    `printf("hello, world\n");`
}

**Write a program** that,
given another program as its argument,
returns TRUE if that argument prints
"`Hello, World!`"

**?????**

# Also Undecidable …

Seems like no algorithm can compute
**anything** about
the language of a Turing Machine,
i.e., about the runtime behavior of programs …

- $REGULAR_{\mathrm{TM}}$ = {<$M$> | $M$ is a TM and $L(M)$ is a regular language}
- $CONTEXTFREE_{\mathrm{TM}}$ = {<$M$> | $M$ is a TM and $L(M)$ is a CFL}
- $DECIDABLE_{\mathrm{TM}}$ = {<$M$> | $M$ is a TM and $L(M)$ is a decidable language}
- $FINITE_{\mathrm{TM}}$ = {<$M$> | $M$ is a TM and $L(M)$ is a finite language}
- …

Rice's Theorem

- $ANYTHING_{\mathrm{TM}}$ = {<$M$> | $M$ is a TM and "… **anything** …" about $L(M)$}

# Rice's Theorem: $ANYTHING_{\text{TM}}$ is Undecidable

$ANYTHING_{\text{TM}}$ = {<$M$> | $M$ is a TM and … **anything** … about $L(M)$}

- "… **Anything** …", more precisely:

  For any $M_1$, $M_2$,

  - if $L(M_1) = L(M_2)$
  - then $M_1 \in ANYTHING_{\text{TM}} \Leftrightarrow M_2 \in ANYTHING_{\text{TM}}$

- Also, "… **Anything** …"must be "non-trivial":
  - $ANYTHING_{\text{TM}}$ != {}
  - $ANYTHING_{\text{TM}}$ != set of all TMs

# Rice's Theorem: $ANYTHING_{TM}$ is Undecidable

$ANYTHING_{TM}$ = {<$M$> | $M$ is a TM and … **anything** … about $L(M)$}

Proof by **contradiction**

- Assume some language satisfying $ANYTHING_{TM}$ has a decider $R$.
  - Since $ANYTHING_{TM}$ is non-trivial, then there exists $M_{ANY} \in ANYTHING_{TM}$
  - Where $R$ accepts $M_{ANY}$
- Use $R$ to create decider for $A_{TM}$:

On input <$M$, $w$>:

- Create $M_w$:

  $M_w$ = on input $x$:
  - Run $M$ on $w$
    - If $M$ rejects $w$: reject $x$
    - If $M$ accepts $w$:
      Run $M_{ANY}$ on $x$ and accept if it accepts, else reject

  If $M$ accepts $w$: $M_w = M_{\text{ANY}}$
  If $M$ doesn't accept $w$: $M_w$ accepts nothing

- Run $R$ on $M_w$
  - If it accepts, then $M_w = M_{ANY}$, so $M$ accepts $w$, so accept
  - Else reject

These two cases must be different, (so $R$ can distinguish when $M$ accepts $w$)

**Wait**! What if the TM that accepts nothing is in $ANYTHING_{TM}$!

Proof still works! Just use the complement of $ANYTHING_{TM}$ instead!

# Rice’s Theorem Implication

{<$M$> | $M$ is a TM that installs malware}

**Undecidable!**
(by Rice’s Theorem)