UMB CS 622

Undecidability

Friday, April 19, 2024

Diagram showing inclusion relationships among Turing-recognizable, decidable, context-free, and regular languages.
Announcements

• HW 9 out
  • due Wednesday 4/22 12pm noon

Lecture Participation Question
• Is the Universal Turing Machine ($A_{TM}$) a decider? A recognizer?
Recap: Decidability of Regular and CFLs

- \(A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w \}\) Decidable
- \(A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts input string } w \}\) Decidable
- \(A_{REX} = \{ \langle R, w \rangle \mid R \text{ is a regular expression that generates string } w \}\) Decidable
- \(E_{DFA} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset \}\) Decidable
- \(EQ_{DFA} = \{ \langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \}\) Decidable
- \(A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \}\) Decidable
- \(E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset \}\) Decidable
- \(EQ_{CFG} = \{ \langle G, H \rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H) \}\) Undecidable?
- \(A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}\) Undecidable?
Thm: $A_{TM}$ is Turing-recognizable

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

$U =$ “On input $\langle M, w \rangle$, where $M$ is a TM and $w$ is a string:
1. Simulate $M$ on input $w$.
2. If $M$ ever enters its accept state, accept; if $M$ ever enters its reject state, reject.”

$U =$ Implements TM computation steps
- i.e., “The Universal Turing Machine”
- “Program” simulating other programs (interpreter)
- **Problem:** $U$ loops when $M$ loops

Termination argument?

So it’s a recognizer, not a decider
**Thm:** \( A_{TM} \) is Turing-recognizable

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

\( U = \) “On input \( \langle M, w \rangle \), where \( M \) is a TM and \( w \) is a string:

1. Simulate \( M \) on input \( w \).
2. If \( M \) ever enters its accept state, accept; if \( M \) ever enters its reject state, reject.”

<table>
<thead>
<tr>
<th>Example Str</th>
<th>( M ) on input ( w )?</th>
<th>( U )?</th>
<th>In ( A_{TM} ) lang?</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle M_1, 01#01 \rangle )</td>
<td>Accept</td>
<td>Accept</td>
<td>Yes</td>
</tr>
<tr>
<td>( \langle M_1, 00#11 \rangle )</td>
<td>Reject</td>
<td>Reject</td>
<td>No</td>
</tr>
<tr>
<td>( \langle M_{loop}, * \rangle )</td>
<td>Loop!</td>
<td>Loop!</td>
<td>No</td>
</tr>
</tbody>
</table>

Let:
- \( M_1 = \) “\( w\#w \)” lang decider
- \( M_{loop} = \) looping TM

Columns must match!

Is this right? Yes!
**Thm:** $A_{TM}$ is undecidable

\[ A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ accepts } w \} \]
Prove: Spider-Man does not exist

In general, proving something not true is different (and harder) than proving it true.

In some cases, it’s possible, but typically requires new proof techniques!

Example (Regular Languages)
Prove a language is regular:
- Create a DFA
Prove a language is not regular:
- Proof by contradiction using Pumping Lemma
Thm: $A_{TM}$ is undecidable

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

Example (decidable languages)

Prove a language is **decidable**:
- Create a decider TM (with termination argument)

Prove a language is **not decidable**:
- ????
Kinds of Functions (a fn maps Domain → Range)

• **Injective**, a.k.a., “one-to-one”
  • Every element in Domain has a unique mapping
  • How to remember:
    • Entire Domain is mapped “in” to the Range

• **Surjective**, a.k.a., “onto”
  • Every element in Range is mapped to
  • How to remember:
    • “Sur” = “over” (eg, survey); Domain is mapped “over” the Range

• **Bijective**, a.k.a., “correspondence” or “one-to-one correspondence”
  • Is both injective and surjective
  • Unique pairing of every element in Domain and Range
Countability

• A set is “countable” if it is:
  • Finite
  • Or, there exists a bijection between the set and the natural numbers
    • In this case, the set has the same size as the set of natural numbers
    • This is called “countably infinite”
Exercise: Which set is larger?

- The set of:
  - Natural numbers, or
  - Even numbers?

- They are the same size! Both are countably infinite
  - Proof: Bijection:

<table>
<thead>
<tr>
<th>$n$</th>
<th>$f(n) = 2n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Every natural number maps to a unique even number, and vice versa.
Exercise: Which set is larger?

- The set of:
  - Natural numbers $\mathbb{N}$, or
  - Positive rational numbers? $\mathbb{Q} = \{ \frac{m}{n} \mid m, n \in \mathbb{N} \}$
- They are the same size! Both are countably infinite

A possible mapping of Natural numbers to Positive rationals?

So these don’t get mapped to: (not a bijection)

But, each row is infinite

Positive rational numbers
Exercise: Which set is larger?

- The set of:
  - Natural numbers \( \mathcal{N} \), or
  - Positive rational numbers? \( Q = \{ \frac{m}{n} \mid m, n \in \mathcal{N} \} \)

- They are the same size! Both are countably infinite.

Another mapping: This is a bijection bc every natural number maps to a unique fraction, and vice versa.
Exercise: Which set is larger?

• The set of:
  • Natural numbers $\mathbb{N}$, or
  • Real numbers $\mathbb{R}$
• There are more real numbers. It is uncountably infinite.

Proof, by contradiction:
• Assume a bijection between natural and real numbers exists.
  • So: every natural num maps to a unique real, and vice versa
But we show that in any given mapping,
  • Some real number is not mapped to ...
  • E.g., a number that has different digits at each position:

\[ x = 0.4641 \ldots \]

• This number cannot be in the mapping ...
• ... So we have a contradiction!
Georg Cantor

• Invented set theory

• Came up with countable infinity (1873)

• And uncountability:
  • Also: how to show uncountability with “diagonalization” technique
Diagonalization with Turing Machines

**Diagonal: Result of Giving a TM its own Encoding as Input**

<table>
<thead>
<tr>
<th></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>(\langle D \rangle)</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M_1)</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>...</td>
</tr>
<tr>
<td>(M_2)</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>...</td>
</tr>
<tr>
<td>(M_3)</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>...</td>
</tr>
<tr>
<td>(M_4)</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(D)</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>?</td>
</tr>
</tbody>
</table>

Try to construct "opposite" TM \(D\)

**opposites**

**All TMs**

**What should happen here?**

**It must both accept and reject!**

**TM \(D\) can’t exist!**
Thm: \( A_{TM} \) is undecidable

Proof by contradiction:

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

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

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.”

From previous slide (does opposite of what input TM would do if given itself)

\( H \) computes \( M \)’s result with itself as input

Do the opposite
Thm: $A_{TM}$ is undecidable

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

Proof by contradiction:

1. Assume $A_{TM}$ is decidable. So there exists a decider $H$ for it:
   
   $H(\langle M, w \rangle) = \begin{cases} 
   \text{accept} & \text{if } M \text{ accepts } w \\
   \text{reject} & \text{if } M \text{ does not accept } w 
   \end{cases}$

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. But $D$ does not exist! Contradiction! So the assumption is false.
Easier Undecidability Proofs

• We proved \( A_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \} \) undecidable ...

• ... by contradiction:
  • Use hypothetical \( A_{\text{TM}} \) decider to create an impossible decider “\( D \)”!

  reduce “\( D \) problem” to \( A_{\text{TM}} \)

• Step # 1: coming up with “\( D \)” --- hard!
  • Need to invent diagonalization

• Step # 2: reduce “\( D \)” problem to \( A_{\text{TM}} \) --- easier!

• From now on: undecidability proofs only need step # 2!
  • And we now have two “impossible” problems to choose from
The Halting Problem

\[ \text{HALT}_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w \} \]

**Thm**: \( \text{HALT}_{TM} \) is undecidable

**Proof**, by contradiction:

- **Assume**: \( \text{HALT}_{TM} \) has **decider** \( R \); use it to create decider for \( A_{TM} \):
  \[ A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ accepts } w \} \]

- ...

- But \( A_{TM} \) is undecidable and has no decider!
The Halting Problem

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

**Proof, by contradiction:**

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

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

$S$ = “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. **This means** $M$ loops on input $w$.
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.”

**Termination argument:**

**Step 1:** $R$ is a decider so always halts

**Step 3:** $M$ always halts because $R$ said so
The Halting Problem

Thm: \( \text{HALT}_\text{TM} \) is undecidable

Proof, by contradiction:

- **Assume:** \( \text{HALT}_\text{TM} \) has **decider** \( R \); use it to create decider for \( A_\text{TM} \):
  \[
  A_\text{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \} 
  \]

  \[
  S = \text{“On input } \langle M, w \rangle, \text{ an encoding of a TM } M \text{ and a string } w:} 
  \]
  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.”

- But \( A_\text{TM} \) is undecidable! i.e., this decider does not exist!
  - So \( \text{HALT}_\text{TM} \) is also undecidable!
Interlude: Reducing from $\text{HALT}_\text{TM}$

A practical thought experiment ... ... about compiler optimizations

Your compiler changes your program!

If TRUE then A else B $\rightarrow$ A

$1 + 2 + 3 \rightarrow 6$
Compiler Optimizations

Optimization - docs

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

**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 (accepts $<M,w>$ if $M$ halts with $w$, else rejects):

$S = \text{On input }<M, w>$, where $M$ is C++ program and $w$ is string:

- If $OPT(M) = \text{for}(;;)$
  - 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

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

Similar languages

It's straightforward to use hypothetical \( HALT_{TM} \) decider to create \( A_{TM} \) decider
Summary: The Limits of Algorithms

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

Decidable
Decidable
Undecidable
Undecidable
Decidable
Decidable
Undecidable

How can we use a hypothetical $E_{TM}$ decider to create $A_{TM}$ or $HALT_{TM}$ decider?
Reducibility: Modifying the TM

**Thm:** $E_{TM}$ is undecidable

**Proof, by contradiction:**

- Assume $E_{TM}$ has decider $R$; use it to create decider for $A_{TM}$:
  
  $S = \text{"On input } \langle M, w \rangle, \text{ an encoding of a TM } M \text{ and a string } w:\$

  - **First, construct $M_1$**
    
    - Run $R$ on input $\langle M \rangle$
    - 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 accept $w$:
    
    $M_1 = \text{"On input } x:\$
    
    1. If $x \neq w$, reject.
    2. If $x = w$, run $M$ on input $w$ and accept if $M$ does.

  Input is $w$, maybe accept
  
  Input not $w$, always reject
  
  $M_1$ accepts $w$ if $M$ does
Reducibility: Modifying the TM

**Thm:** $E_{TM}$ is undecidable

**Proof**, by contradiction:

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

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

  1. Run $R$ on input $\langle M \rangle$
  2. If $R$ accepts, reject (because it means $\langle M \rangle$ doesn’t accept $w$)
  3. If $R$ rejects, then accept ($\langle M \rangle$ accepts $w$)

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

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

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

This decider for $A_{TM}$ cannot exist!
Summary: The Limits of Algorithms

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

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

Proof, by contradiction:

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

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

  $S = \text{“On input } \langle M \rangle, \text{ where } M \text{ 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: $EQ_{TM}$ is undecidable

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

Proof, by contradiction:

• Assume: $EQ_{TM}$ has decider $R$; use it to create decider for $E_{TM}$:

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

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 $E_{TM}$ is undecidable!
Summary: Undecidability Proof Techniques

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

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

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

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

\[ REGULAR_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language} \} \]
Thm: $\text{REGULAR}_{\text{TM}}$ is undecidable

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

Proof, by contradiction:

- **Assume:** $\text{REGULAR}_{\text{TM}}$ has *decider* $R$; use it to create *decider* for $A_{\text{TM}}$:

  $S = \text{“On input } \langle M, w \rangle, \text{ an encoding of a TM } M \text{ 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 $w$
Thm: $REGULAR_{TM}$ is undecidable (continued)

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

$M_2 = \text{“On input } x:\text{ }
1. \text{ If } x \text{ has the form } 0^n1^n, \text{ accept.}
2. \text{ If } x \text{ does not have this form, run } M \text{ on input } w \text{ and accept if } M \text{ accepts } w.”$

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

if $M$ does not accept $w$, $M_2$ accepts all strings (regular lang)

Always accept strings $0^n1^n$
$L(M_2) = \text{nonregular, so far}$

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

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

188
Also Undecidable ...

- $\text{REGULAR}_{\text{TM}} = \{<M> | M \text{ is a TM and } L(M) \text{ is a regular language}\}$

- $\text{CONTEXTFREE}_{\text{TM}} = \{<M> | M \text{ is a TM and } L(M) \text{ is a CFL}\}$

- $\text{DECIDABLE}_{\text{TM}} = \{<M> | M \text{ is a TM and } L(M) \text{ is a decidable language}\}$

- $\text{FINITE}_{\text{TM}} = \{<M> | M \text{ is a TM and } L(M) \text{ is a finite language}\}$

Seems like no algorithm can compute anything about the language of a Turing Machine, i.e., about the runtime behavior of programs ...
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
Write a program that, given another program as its argument, returns `TRUE` if that argument prints “Hello, World!”

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

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

- $\text{REGULAR}_{TM} = \{<M> | M \text{ is a TM and } L(M) \text{ is a regular language}\}$

- $\text{CONTEXTFREE}_{TM} = \{<M> | M \text{ is a TM and } L(M) \text{ is a CFL}\}$

- $\text{DECIDABLE}_{TM} = \{<M> | M \text{ is a TM and } L(M) \text{ is a decidable language}\}$

- $\text{FINITE}_{TM} = \{<M> | M \text{ is a TM and } L(M) \text{ is a finite language}\}$

- ...

- $\text{ANYTHING}_{TM} = \{<M> | M \text{ is a TM and “… anything …” about } L(M)\}$
Rice’s Theorem: $\text{ANYTHING}_{TM}$ is Undecidable

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

• “... Anything ...”, more precisely:
  For any $M_1, M_2$,
  • if $L(M_1) = L(M_2)$
  • then $M_1 \in \text{ANYTHING}_{TM} \iff M_2 \in \text{ANYTHING}_{TM}$

• Also, “... Anything ...” must be “non-trivial”:
  • $\text{ANYTHING}_{TM} \neq \{\}$
  • $\text{ANYTHING}_{TM} \neq \text{set of all TMs}$
Rice’s Theorem: $\text{ANYTHING}_{\text{TM}}$ is Undecidable

$\text{ANYTHING}_{\text{TM}} = \{<M> \mid M \text{ is a TM and … anything … about } L(M)\}$

Proof by contradiction

• **Assume** some language satisfying $\text{ANYTHING}_{\text{TM}}$ has a decider $R$.
  • Since $\text{ANYTHING}_{\text{TM}}$ is non-trivial, then there exists $M_{\text{ANY}} \in \text{ANYTHING}_{\text{TM}}$
  • Where $R$ accepts $M_{\text{ANY}}$

• Use $R$ to create decider for $A_{\text{TM}}$:

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

  • **Create** $M_w$:
    - Run $M$ on $w$
      - If $M$ rejects $w$: reject $x$
      - If $M$ accepts $w$:
        Run $M_{\text{ANY}}$ 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 $M_w = M_{\text{ANY}}$, so $M$ accepts $w$, so accepts nothing
    - 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 $\text{ANYTHING}_{\text{TM}}$!

Proof still works! Just use the complement of $\text{ANYTHING}_{\text{TM}}$ instead!
Rice’s Theorem Implication

\{\langle M \rangle \mid M \text{ is a TM that installs malware}\}

Undecidable!
(by Rice’s Theorem)
Submit in-class participation question

On gradescope