Today's learning goals

- State and use the Church-Turing thesis.
- Give examples of decidable problems.
- Review for exam 2
  - Wednesday May 11 8pm-10pm
  - Bring photo ID, 3inch by 5inch note card, seat assignment
  - Office hours available
Deciders and recognizers  

- $L$ is **Turing-recognizable** if some Turing machine recognizes it.

- $M$ is a **decider** TM if it halts on all inputs.

- $L$ is **Turing-decidable** if some Turing machine that is a decider recognizes it.
Variants of TMs

- Scratch work, copy input, …
- Parallel computation
- Printing vs. accepting
- More flexible transition function
  - Can "stay put"
  - Can "get stuck"
    - lots of examples in exercises to Chapter 3

Also: wildly different models

- λ-calculus, Post canonical systems, URMs, etc.
Algorithm

- Wikipedia "self-contained step-by-step set of operations to be performed"
- CSE 20 textbook "An algorithm is a finite sequence of precise instructions for performing a computation or for solving a problem."

Church-Turing thesis

Each algorithm can be implemented by some Turing machine.
Encoding input for TMs

• By definition, TM inputs are **strings**

• To define TM M:

> "On input w ..."

  1. ...
  2. ...
  3. ...

For inputs that aren't strings, we have to **encode the object** (represent it as a string) first

**Notation:**

- \(<O>\) is the string that represents **(encodes)** the object \(O\)
- \(<O_1, ..., O_n>\) is the **single** string that represents the tuple of objects \(O_1, ..., O_n\)
- \(<...>\) means .... . toString()
Encoding input for TMs

- Payoff: problems we care about can be reframed as languages of strings

  e.g. "Recognize whether a string is a palindrome."
  \[ \{ w \mid w \text{ in } \{0,1\}^* \text{ and } w = w^R \} \]

  e.g. "Recognize Pythagorean triples."
  \[ \{ <a,b,c> \mid a,b,c \text{ integers and } a^2 + b^2 = c^2 \} \]

  e.g. "Check whether a string is accepted by a DFA."
  \[ \{ <B,w> \mid B \text{ is a DFA over } \Sigma, w \text{ in } \Sigma^*, \text{ and } w \text{ is in } L(B) \} \]

  e.g. "Check whether the language of a PDA is infinite."
  \[ \{ <A> \mid A \text{ is a PDA and } L(A) \text{ is infinite} \} \]
Computational problems

A computational problem is **decidable** iff the language encoding the problem instances is decidable.
Computational problems

Sample computational problems and their encodings:

- $A_{\text{DFA}}$ "Check whether a string is accepted by a DFA."
  \[
  \{ <B, w> \mid B \text{ is a DFA over } \Sigma, \text{ w in } \Sigma^*, \text{ and w is in } L(B) \}
  \]

- $E_{\text{DFA}}$ "Check whether the language of a DFA is empty."
  \[
  \{ <A> \mid A \text{ is a DFA over } \Sigma, \text{ L(A) is empty} \}
  \]

- $E_{\text{Q, DFA}}$ "Check whether the languages of two DFA are equal."
  \[
  \{ <A, B> \mid A \text{ and B are DFA over } \Sigma, \text{ L(A) = L(B)} \}
  \]

**FACT:** all of these problems are decidable!
Proving decidability

Claim: $A_{\text{DFA}}$ is decidable

Proof: WTS that \{ $<B,w>$ | $B$ is a DFA over $\Sigma$, $w$ in $\Sigma^*$, and $w$ is in $L(B)$ \} is decidable.

Step 1: construction

How would you check if $w$ is in $L(B)$?
Proving decidability

Claim: \( A_{DFA} \) is decidable

Proof: WTS that \( \{ <B,w> \mid B \text{ is a DFA over } \Sigma, w \in \Sigma^*, \text{ and } w \text{ is in } L(B) \} \) is decidable.

Step 1: construction

Define TM \( M \) by: \( M_1 = \) "On input \( <B,w> \):

1. Check whether \( B \) is a valid encoding of a DFA and \( w \) is a valid input for \( B \). If not, reject.
2. Simulate running \( B \) on \( w \) (by keeping track of states in \( B \), transition function of \( B \), etc.)
3. When the simulation ends, by finishing to process all of \( w \), check current state of \( B \): if it is final, accept; if it is not, reject."

What kind of construction is this?

A. Formal definition of TM
B. Implementation-level description of TM
C. High-level description of TM
D. I don't know.
Proving decidability

Step 1: construction
Define TM $M_1$ by $M_1 = \text{"On input } \langle B, w \rangle \text{"}
1. Check whether $B$ is a valid encoding of a DFA and $w$ is a valid input for $B$. If not, reject.
2. Simulate running $B$ on $w$ (by keeping track of states in $B$, transition function of $B$, etc.)
3. When the simulation ends, by finishing to process all of $w$, check current state of $B$: if it is final, accept; if it is not, reject.

Step 2: correctness proof
WTS (1) $L(M_1) = A_{DFA}$ and (2) $M_1$ is a decider.
Proving decidability

Claim: $E_{\text{DFA}}$ is decidable

Proof: WTS that $\{ <A> | A \text{ is a DFA over } \Sigma, L(A) \text{ is empty } \}$ is decidable.

Idea: give high-level description

Step 1: construction

*What condition distinguishes between DFA that accept *some* string and those that don't accept *any*?*

A. $<A>$ is in $E_{\text{DFA}}$ iff A's initial state is accepting.
B. $<A>$ is in $E_{\text{DFA}}$ iff A' set of accepting states is empty.
C. $<A>$ is in $E_{\text{DFA}}$ iff A is the empty set.
D. None of the above.
E. I don't know.
Proving decidability

**Claim:** $E_{DFA}$ is decidable

**Proof:** WTS that \{ $<A>$ | $A$ is a DFA over $\Sigma$, $L(A)$ is empty \} is decidable.

Idea: give high-level description

Step 1: construction

What condition distinguishes between DFA that accept *some* string and those that don't accept *any*?
Claim: \( E_{\text{DFA}} \) is decidable

Proof: WTS that \( \{ <A> | A \text{ is a DFA over } \Sigma, L(A) \text{ is empty} \} \) is decidable. Idea: give high-level description

Step 1: construction

Define TM \( M_2 \) by: \( M_2 = \) "On input \( <A> \):

1. Check whether \( A \) is a valid encoding of a DFA; if not, reject.
2. Mark the start state of \( A \).
3. Repeat until no new states get marked:
   i. Loop over states of \( A \) and mark any unmarked state that has an incoming edge from a marked state.
4. If no final state of \( A \) is marked, accept; otherwise, reject."
Proving decidability

Step 1: construction
Define TM $M_2$ by: $M_2 =$

1. Check whether $A$ is a valid encoding of a DFA; if not, reject.
2. Mark the state state of $A$.
3. Repeat until no new states get marked:
   i. Loop over states of $A$ and mark any unmarked state that has an incoming edge from a marked state.
4. If no final state of $A$ is marked, accept; otherwise, reject.

Step 2: correctness proof
WTS (1) $L(M_2) = \mathcal{E}_{\text{DFA}}$ and (2) $M_2$ is a decider.

M will mark
A. all the states of the DFA $A$
B. all the states of the DFA $A$ that are reachable from the start state
C. some states in the DFA more than once
D. I don't know.
Proving decidability

**Claim:** $EQ_{\text{DFA}}$ is decidable

**Proof:** WTS that \( \{ <A, B> | A, B \text{ are DFA over } \Sigma, L(A) = L(B) \} \) is decidable. *Idea: give high-level description*

**Step 1:** construction

*Will we be able to simulate A and B?*

*What does set equality mean?*

*Can we use our previous work?*
Proving decidability

Claim: $\text{EQ}_{\text{DFA}}$ is decidable

Proof: WTS that $\{ <A, B> \mid A, B \text{ are DFA over } \Sigma, L(A) = L(B) \}$ is decidable. Idea: give high-level description

Step 1: construction

Will we be able to simulate $A$ and $B$?

What does set equality mean?

Can we use our previous work?

\[ X = Y \text{ iff } ( (X \cap Y^c) \cup (Y \cap X^c) ) = \emptyset \]
Proving decidability

Claim: $\text{EQ}_{\text{DFA}}$ is decidable

Proof: WTS that $\{ <A, B> \mid A, B \text{ are DFA over } \Sigma, L(A) = L(B) \}$ is decidable. Idea: give high-level description

Step 1: construction

Very high-level:

$X = Y \iff ( (X \cap Y^c) \cup (Y \cap X^c) ) = \emptyset$

Build new DFA recognizing symmetric difference of $A, B$. Check if this set is empty.
Claim: $\text{EQ}_{\text{DFA}}$ is decidable

Proof: WTS that $\{ <A, B> | A, B \text{ are DFA over } \Sigma, L(A) = L(B) \}$ is decidable. Idea: give high-level description

Step 1: construction

Define TM $M_3$ by: $M_3 =$ "On input $<A,B>$:
1. Check whether $A, B$ are valid encodings of DFA; if not, reject.
2. Construct a new DFA, $D$, from $A, B$ using algorithms for complementing, taking unions of regular languages such that $L(D) =$ symmetric difference of $A$ and $B$.
3. Run machine $M_2$ on $<D>$.
4. If it accepts, accept; if it rejects, reject."
Proving decidability

Step 1: construction
Define TM $M_3$ by: $M_3 = "On input <A,B>:"
1. Check whether A,B are valid encodings of DFA; if not, reject.
2. Construct a new DFA, D, from A,B using algorithms for complementing, taking unions of regular languages such that $L(D) = \text{symmetric difference of } A \text{ and } B$.
3. Run machine $M_2$ on $<D>$.
4. If it accepts, accept; if it rejects, reject."

Step 2: correctness proof
WTS (1) $L(M_3) = \text{EQ}_{\text{DFA}}$ and (2) $M_3$ is a decider.
Techniques

• **Subroutines**: can use decision procedures of decidable problems as subroutines in other algorithms
  - $A_{DFA}$
  - $E_{DFA}$
  - $EQ_{DFA}$

• **Constructions**: can use algorithms for constructions as subroutines in other algorithms
  - Converting DFA to DFA recognizing complement (or Kleene star).
  - Converting two DFA/NFA to one recognizing union (or intersection, concatenation).
  - Converting NFA to equivalent DFA.
  - Converting regular expression to equivalent NFA.
  - Converting DFA to equivalent regular expression.
Regular
Context-Free
Decidable
Turing-Recognizable
Review: Closure

**Theorem:** The class of recognizable languages over fixed alphabet $\Sigma$ is closed under union.

Proof: Let $L_1$ and $L_2$ be languages and suppose $M_1$ and $M_2$ are TMs deciding these languages. Construct the TM $M$ as "On input $w$.

1. Run $M_1$ on input $w$. If $M_1$ accepts $w$, accept. Otherwise, go to 2.
2. Run $M_2$ on input $w$. If $M_2$ accepts $w$, accept. Otherwise, reject."

Correctness of construction:

WTS $L(M) = L_1 \cup L_2$
Review: Closure

**Theorem**: The class of recognizable languages over fixed alphabet $\Sigma$ is closed under union.

Proof: Let $L_1$ and $L_2$ be languages and suppose $M_1$ and $M_2$ are TMs deciding these languages. Construct the TM $M$ as "On input $w$,

1. Run $M_1$ in parallel with $M_2$ on input $w$.
2. If either accepts, accept. If both reject, reject.

**Correctness of construction:**

WTS $L(M) = L_1 U L_2$
Review: Closure

Run in parallel?

1. Nondeterministically pick either $M_1$ or $M_2$ to run.

2. For integer $i=1$, ...
   i. Run $M_1$ on $w$ for $i$ many steps.
   ii. Run $M_2$ on $w$ for $i$ many steps.
   iii. Increment $i$.  


Review: Closure

**Theorem:** The class of recognizable languages over fixed alphabet $\Sigma$ is closed under concatenation.

Proof: Let $L_1$ and $L_2$ be languages and suppose $M_1$ and $M_2$ are TMs recognizing these languages. "...

**Correctness of construction:**

WTS $L(M) = L_1 \cdot L_2$
Review: Closure

**Theorem:** The class of recognizable languages over fixed alphabet $\Sigma$ is closed under concatenation.

Proof: Let $L_1$ and $L_2$ be languages and suppose $M_1$ and $M_2$ are TMs recognizing these languages.

To build nondeterministic machine $N$: On input $w$:

1. Read $w$ and nondeterministically guess a division $w = xy$.
2. Run $M_1$ on input $x$. If it rejects, reject. If it accepts, go to next step.
3. Run $M_2$ on input $y$. If it rejects, reject. If it accepts, accept.

Correctness of construction:

WTS $L(N) = L_1 \cdot L_2$
Review: Closure

**Theorem:** The class of recognizable languages over fixed alphabet $\Sigma$ is closed under concatenation.

Proof: Let $L_1$ and $L_2$ be languages and suppose $M_1$ and $M_2$ are TMs recognizing these languages.

**To build deterministic machine $M$:** On input $w$:

1. For each positive integer value of $i$:  
   i. For each split $w = xy$
      a. Run $M_1$ on input $x$ for $i$ steps. If it rejects, go to next split. If it accepts, go to step b.
      b. Run $M_2$ on input $y$ for $i$ steps. If it rejects, go to next split. If it accepts, accept.
      c. If none of the splits accepted after $i$ steps of the computation, increment $i$.

**Correctness of construction:**

WTS $L(M) = L_1 L_2$
## Closure properties

<table>
<thead>
<tr>
<th>Closed under ...</th>
<th>Turing-recognizable</th>
<th>Turing-decidable</th>
<th>CFL</th>
<th>Regular</th>
</tr>
</thead>
<tbody>
<tr>
<td>Union</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intersection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Complement</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Star</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Concatenation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Review: CFG

Design a CFG for

\[ L = \{ w \text{ in } \{0,1\}^* \mid w \text{ contains at least three 1s} \} \]

What's the shortest string?

How do you build onto strings?
Design a CFG for
\[ L = \{ w \in \{0,1\}^* \mid w \text{ contains at least three 1s} \} \]

\[ S \rightarrow \text{guarantee that there are 3 1s} \]
\[ R \rightarrow \text{pad between those 1s with any string} \]
Design a CFG for

$L = \{ w \in \{0,1\}^* | w \text{ contains at least three 1s} \}$

$S \rightarrow R1R1R1R$

$R \rightarrow 0R \mid 1R \mid \epsilon$

*Formal definition?*
Review: CFG

What's the language of the CFG

\[ G = ( \{ S, A, B, C \}, \{ a, b, c \}, R, S ) \]

with R

\[
S \rightarrow ABC \\
A \rightarrow aA \mid \varepsilon \\
B \rightarrow bB \mid \varepsilon \\
C \rightarrow cC \mid \varepsilon 
\]
Review: PDA

Give informal English description and state diagram for PDA that recognizes

\[ L = \{ \text{w in } \{0,1\}^* \mid \text{w has more 0s than 1s} \} \]
Review: PDA

Give informal English description and state diagram for PDA that recognizes

\[ L = \{ w \in \{0,1\}^* \mid w \text{ has more 0s than 1s} \} \]

Description:

1. If the input is the empty string, reject.
2. If the stack is empty or if the top symbol matches current input symbol, push input symbol to stack.
3. Otherwise, pop symbol from stack.
4. When finish reading the input: if top of stack is 0, accept; otherwise, reject.

Use stack to record "excess" of one bit or another