Today's learning goals

- State and use the Church-Turing thesis.
- Describe several variants of Turing machines and informally explain why they are equally expressive.
- Explain what it means for a problem to be decidable.
- Justify the use of encoding.
- Give examples of decidable problems.
Consider the TM $M = \"On input $w$,\
1. Run $M_1$ on $w$. If $M_1$ rejects, rejects. If $M_1$ accepts, go to 2.
2. Run $M_2$ on $w$. If $M_2$ accepts, accept. If $M_2$ rejects, 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.

What's $L(M)$?
Is $M$ a decider?
Describing TMs

- **Formal definition**: set of states, input alphabet, tape alphabet, transition function, state state, accept state, reject state.

- **Implementation-level definition**: English prose to describe Turing machine head movements relative to contents of tape.

- **High-level description**: Description of algorithm, without implementation details of machine. As part of this description, can "call" and run another TM as a subroutine.
Subroutines

Consider the TM \( M \) = "On input \( w \),

1. Run \( M_1 \) on \( w \). If \( M_1 \) rejects, rejects. If \( M_1 \) accepts, go to 2.
2. Run \( M_2 \) on \( w \). If \( M_2 \) accepts, accept. If \( M_2 \) rejects, reject."

Claim: \( M \) decides \( L_1 \cap L_2 \)

\[
\begin{align*}
&\text{If } M \text{ decides } L, \\
&M_2 \quad L_2
\end{align*}
\]

\[
\begin{align*}
&M \text{ decides } L_1 \cap L_2
\end{align*}
\]
will not work for recognizing M(x). If it accepts, then accept.

M(x):

\[ M(x) \text{ runs } \text{M}_1 \text{ or } \text{M}_2 \text{ for } x \in \Sigma \]

if M(x) loops, return accept.

\[ \text{Run } \text{M}_1 \text{ or } \text{M}_2 \text{ as specified. If it accepts, accept. Otherwise, return reject.} \]

\[ \text{if either accepts, accept.} \]

recognize L, L'
High-level description = 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."

Each algorithm can be implemented by some Turing machine.

Church-Turing thesis
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

Multiple tapes
Nondeterminism
Enumerators

All these models are equally expressive!

Also: wildly different models
• $\lambda$-calculus, Post canonical systems, URMs, etc.
"Equally expressive"

Model 1 is **equally expressive** as Model 2 iff
- every language recognized by some machine in Model 1 is recognizable by some machine in Model 2, **and**
- every language recognized by some machine in Model 2 is recognizable by some machine in Model 1.

e.g. DFA, NFA equally expressive
Nondeterministic TMs

Transition function

\[ Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L,R\}) \]

Sketch of proof of equivalence:

*To simulate nondeterministic machine:* Use 3 tapes: "read-only" input tape, simulation tape, tape tracking nondeterministic branching.
Multitape TMs

- As part of construction of machine, declare some finite number of tapes that are available.
- Input given on tape 1, rest of the tapes start blank.
- Each tape has its own read/write head.
- Transition function
  \[ Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k \]

**Sketch of proof of equivalence:**

*To simulate multiple tapes with one tape:* Use delimiter to keep tape contents separate, use special symbol to indicate location of each read/write head.
Very different model: Enumerators

Produce language as output rather than recognize input

Finite State Control

Unlimited work tape

Printer

Computation proceeds according to transition function.

At any point, machine may "send" a string to printer.

$L(E) = \{ w \mid E \text{ eventually, in finite time, prints } w \}$
Set of all strings

"For each $\Sigma$, there is an enumerator whose language is the set of all strings over $\Sigma$.

**Proof:** High level description of $E$
Theorem: A language $L$ is Turing-recognizable iff some enumerator enumerates $L$.

Proof:

1. Assume $L$ is Turing-recognizable. WTS some enumerator enumerates it.

2. Assume $L$ is enumerated by some enumerator. WTS $L$ is Turing-recognizable.
Assume the enumerator E enumerates L. WTS L is Turing-recognizable.

We'll use E in a subroutine for high-level description of Turing machine M that will recognize L.
Define M as follows: M = "On input w,
1. Run E. Every time E prints a string, compare it to w.
2. If w ever appears as the output of E, accept."

Correctness?
Assume $L$ is Turing-recognizable. WTS some enumerator enumerates it.

Let $M$ be a TM that recognizes $L$. We'll use $M$ in a subroutine for high-level description of enumerator $E$.

Let $s_1, s_2, \ldots$ be a list of all strings in $\Sigma^*$. Define $E$ as follows:

$E = "\text{Repeat the following for each value of } i=1,2,3\ldots$

1. Run $M$ for $i$ steps on each input $s_1, \ldots, s_i$
2. If any of the $i$ computations of $M$ accepts, print out the accepted string."

Correctness?
<table>
<thead>
<tr>
<th>M is TM that recognizes L</th>
<th>D is TM that decides L</th>
<th>E is enumerator that enumerates L</th>
</tr>
</thead>
<tbody>
<tr>
<td>If string $w$ is in $L$</td>
<td>accept</td>
<td>accept</td>
</tr>
<tr>
<td>then $\ldots$</td>
<td></td>
<td>print $w$ at some point</td>
</tr>
<tr>
<td>If string $w$ is not in</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>$L$ then $\ldots$</td>
<td></td>
<td>never print $w$</td>
</tr>
<tr>
<td></td>
<td>or loop</td>
<td></td>
</tr>
</tbody>
</table>
At start of CSE 105…

- Pick a model of computation
- Study what problems it can solve
- Prove its limits

**Classification**: is input of type A or not?

**Decision problem**

\{ w | w is of type A \}

PRIME = \{ 2, 3, 5, 7, … \}

SORTED = \{ <1,3>, <-1, 8, 17> … \}

Decision problems are coded by sets of strings
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 $O_1, ..., O_n$
Encoding inputs

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

e.g. "Recognize whether a string is a palindrome."
    \[ \{ w \mid w \in \{0,1\}^* \text{ and } w = w^R \} \]
e.g. "Check whether a string is accepted by a DFA."
    \[ \{ <B,w> \mid B \text{ is a DFA over } \Sigma, w \in \Sigma^*, \text{ and } w \text{ is in } L(B) \} \subset \{0,1\}^* \]
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_{DFA}** "Check whether a string is accepted by a DFA."
  \{ <B,w> | B is a DFA over Σ, w in Σ*, and \( w \) is in L(B) \}

- **E_{DFA}** "Check whether the language of a DFA is empty."
  \{ <A> | A is a DFA over Σ, L(A) is empty \}

- **EQ_{DFA}** "Check whether the languages of two DFAs are equal."
  \{ <A, B> | A and B are DFAs over Σ, L(A) = L(B) \}

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

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

Define TM $M_1$ by: $M_1 = \text{"On input } <B,w>\text{"}$

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."
Proving decidability

Step 1: construction
Define TM $M_1$ by $M_1 = \text{"On input } <B,w>\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 $\{ \langle A \rangle \mid A$ is a DFA over $\Sigma$, $L(A)$ is empty $\}$ is decidable.
Proving decidability

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

e.g. $<$ is in $E_{DFA}$; $<$

$>$ is not in $E_{DFA}$

TM deciding $E_{DFA}$ should accept and should reject
Proving decidability

Claim: $E_{\text{DFA}}$ 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*?
Proving decidability

Claim: $E_{DFA}$ is decidable

Proof: WTS that $\{ <A> \mid A \text{ is a DFA over } \Sigma, \text{ } L(A) \text{ is empty} \}$ 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*?
Proving decidability

Claim: \( \text{E}_{\text{DFA}} \) 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 = \text{"On input } \langle A \rangle:\$
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) $L(M_2) = E_{DFA}$ and (2) $M_2$ is a decider.
Proving decidability

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

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

Step 1: construction

Will we be able to simulate $X$ and $Y$?

What does set equality mean?

Can we use our previous work?
Proving decidability

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

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

Step 1: construction

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

Very high-level:
Build new DFA recognizing symmetric difference of A, B. Check if this set is empty.
Proving decidability

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

Proof: WTS that $\{ <A, B> \mid A, B$ are DFAs 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.