Turing Machines-Synopsis

• The most general model of computation
• Computations of a TM are described by a sequence of configurations. (Accepting Configuration, Rejecting Configuration)
• Turing-recognizable languages
  • TM halts in an accepting configuration if $w$ is in the language.
  • TM may halt in a rejecting configuration or go on indefinitely if $w$ is not in the language.
• Turing-decidable languages
  • TM halts in an accepting configuration if $w$ is in the language.
  • TM halts in a rejecting configuration if $w$ is not in the language.
• Nondeterministic TMs are equivalent to Deterministic TMs.
For the rest of the course we will have a rather standard way of describing TMs and their inputs. The inputs to TMs have to be strings. Every object $O$ that enters a computation will be represented with a string $\langle O \rangle$, encoding the object. For example if $G$ is a 4 node undirected graph with 4 edges $\langle G \rangle = (1, 2, 3, 4) \ ( (1, 2) , \ (2, 3) , \ (3, 1) , \ (1, 4) )$ Then we can define problems over graphs, e.g., as:

$$A = \{ \langle G \rangle \ | \ G \text{ is a connected undirected graph} \}$$
Decidability

- We investigate the power of algorithms to solve problems.
- We discuss certain problems that can be solved algorithmically and others that can not be.
- Why discuss unsolvability?
- Knowing a problem is unsolvable is useful because
  - you realize it must be simplified or altered before you find an algorithmic solution.
  - you gain a better perspective on computation and its limitations.
Decidable Languages
Diagonalization
Halting Problem as a undecidable problem
Turing-unrecognizable languages.
Decidable Languages

Some notational details

- $\langle B \rangle$ represents the encoding of the description of an automaton (DFA/NFA).
- We need to encode $Q, \Sigma, \delta$ and $F$. 
Here is one possible encoding scheme:

Encode $Q$ using unary encoding:
- For $Q = \{q_0, q_1, \ldots q_{n-1}\}$, encode $q_i$ using $i + 1$ 0's, i.e., using the string $0^{i+1}$.
- We assume that $q_0$ is always the start state.

Encode $\Sigma$ using unary encoding:
- For $\Sigma = \{a_1, a_2, \ldots a_m\}$, encode $a_i$ using $i$ 0's, i.e., using the string $0^i$.

With these conventions, all we need to encode is $\delta$ and $F$!

Each entry of $\delta$, e.g., $\delta(q_i, a_j) = q_k$ is encoded as

$$0^{i+1} q_i 1 0^j a_j 1 0^{k+1} q_k$$
The whole $\delta$ can now be encoded as

$$00100001000 \underbrace{1 000001001000000}_\text{transition}_1 \cdots 1 000000100000010_\text{transition}_t$$

$F$ can be encoded just as a list of the encodings of all the final states. For example, if states 2 and 4 are the final states, $F$ could be encoded as

$$000 \underbrace{1 00000}_\text{encoding of the final states}$$

The whole DFA would be encoded by

$$11 00100010000100000 \cdots 0 11 000000001000000011$$

encoding of the transitions

encoding of the final states
Encoding Finite Automata as Strings

- $\langle B \rangle$ representing the encoding of the description of an automaton (DFA/NFA) would be something like

$$\langle B \rangle = 11\underbrace{00100010000100000}_1 \cdots 011\underbrace{0000000010000000}_{11}11$$

encoding of the transitions

encoding of the final states

- In fact, the description of all DFAs could be described by a regular expression like

$$11(0^+10^+10^+1)^*1(0^+1)^+1$$

- Similarly strings over $\Sigma$ can be encoded with (the same convention)

$$\langle w \rangle = 0000\underbrace{10000001}_a \cdots 0$$
Encoding Finite Automata as Strings

- \( \langle B, w \rangle \) represents the encoding of a machine followed by an input string, in the manner above (with a suitable separator between \( \langle B \rangle \) and \( \langle w \rangle \).

- Now we can describe our problems over languages and automata as problems over strings (representing automata and languages).
Decidable Problems

Regular Languages

- Does $B$ accept $w$?
- Is $L(B)$ empty?
- Is $L(A) = L(B)$?
The Acceptance Problem for DFAs

Theorem 4.1

\[ A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w \} \] is a decidable language.

Proof

- Simulate with a two-tape TM.
  - One tape has \( \langle B, w \rangle \)
  - The other tape is a work tape that keeps track of which state of \( B \) the simulation is in.

- \( M = \) “On input \( \langle B, w \rangle \)
  1. Simulate \( B \) on input \( w \)
  2. If the simulation ends in an accept state of \( B \), accept; if it ends in a nonaccepting state, reject.”
The Acceptance Problem for NFAs

Theorem 4.2

$A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is a NFA that accepts input string } w \}$ is a decidable language.

Proof

- Convert NFA to DFA and use Theorem 4.1
- $N = \text{“On input } \langle B, w \rangle \text{ where } B \text{ is an NFA}$
  1. Convert NFA $B$ to an equivalent DFA $C$, using the determinization procedure.
  2. Run TM $M$ in Thm 4.1 on input $\langle C, w \rangle$
  3. If $M$ accepts accept; otherwise reject.”
**The Generation Problem for Regular Expressions**

**Theorem 4.3**

\[ A_{\text{REX}} = \{ \langle R, w \rangle \mid R \text{ is a regular exp. that generates string } w \} \]

is a decidable language.

**Proof**

- Note \( R \) is already a string!!
- Convert \( R \) to an NFA and use Theorem 4.2
- \( P = \) “On input \( \langle R, w \rangle \) where \( R \) is a regular expression
  1. Convert \( R \) to an equivalent NFA \( A \), using the Regular Expression-to-NFA procedure
  2. Run TM \( N \) in Thm 4.2 on input \( \langle A, w \rangle \)
  3. If \( N \) accepts accept; otherwise reject.”
The Emptiness Problem for DFAs

**Theorem 4.4**

\[ E_{DFA} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset \} \] is a decidable language.

**Proof**

- Use the DFS algorithm to mark the states of DFA
- \( T = \) “On input \( \langle A \rangle \) where \( A \) is a DFA.
  1. Mark the start state of \( A \)
  2. Repeat until no new states get marked.
  - Mark any state that has a transition coming into it from any state already marked.
  3. If no final state is marked, *accept*; otherwise *reject*.\)”
The Equivalence Problem for DFAs

**Theorem 4.5**

$EQ_{DFA} = \{\langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ is a decidable language.

**Proof**

- Construct the machine for $L(C) = (L(A) \cap \overline{L(B)}) \cup (L(A) \cap L(B))$ and check if $L(C) = \Phi$.
- $T =$ "On input $\langle A, B \rangle$ where $A$ and $B$ are DFAs.
  1. Construct the DFA for $L(C)$ as described above.
  2. Run TM $T$ of Theorem 4.4 on input $\langle C \rangle$.
  3. If $T$ accepts, *accept*; otherwise *reject*."
Decidable Problems
Context-free Languages

- Does grammar $G$ generate $w$?
- Is $L(G)$ empty?
THEOREM 4.7

\[ A_{\text{CFG}} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \] is a decidable language.

PROOF

- Convert \( G \) to Chomsky Normal Form and use the CYK algorithm.
- \( C = "\text{On input } \langle G, w \rangle \text{ where } G \text{ is a CFG} \)
  
  1. Convert \( G \) to an equivalent grammar in CNF
  2. Run CYK algorithm on \( w \) of length \( n \)
  3. If \( S \in V_{i,n} \) accept; otherwise reject."
THE GENERATION PROBLEM FOR CFGs

ALTERNATIVE PROOF

- Convert $G$ to Chomsky Normal Form and check all derivations up to a certain length (Why!)
- $S =$ “On input $\langle G, w \rangle$ where $G$ is a CFG
  1. Convert $G$ to an equivalent grammar in CNF
  2. List all derivations with $2n - 1$ steps where $n$ is the length of $w$. If $n = 0$ list all derivations of length 1.
  3. If any of these strings generated is equal to $w$, accept; otherwise reject.”

- This works because every derivation using a CFG in CNF either increase the length of the sentential form by 1 (using a rule like $A \to BC$ or leaves it the same (using a rule like $A \to a$)

- Obviously this is not very efficient as there may be exponentially many strings of length up to $2n - 1$. 
THE EMPTINESS PROBLEM FOR CFGs

THEOREM 4.8

\[ E_{\text{CFG}} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset \} \text{ is a decidable language.} \]

PROOF

- Mark variables of \( G \) systematically if they can generate terminal strings, and check if \( S \) is unmarked.

- \( R = \text{“On input } \langle G \rangle \text{ where } G \text{ is a CFG.} \)
  1. Mark all terminal symbols \( G \)
  2. Repeat until no new variable get marked.
     - Mark any variable \( A \) such that \( G \) has a rule \( A \rightarrow U_1 U_2 \cdots U_k \)
       and \( U_1, U_2, \ldots U_k \) are already marked.
  3. If start symbol is NOT marked, accept; otherwise reject.”
The Equivalence Problem for CFLs

\[ EQ_{CFG} = \{ \langle G, H \rangle \mid G \text{ and } H \text{ are CFLs and } L(G) = L(H) \} \]

- It turns out that \( EQ_{DFA} \) is not a decidable language.
- The construction for DFAs does not work because CFLs are NOT closed under intersection and complementation.
- Proof comes later.
Decidability of CFLs

Theorem 4.9
Every context free language is decidable.

Proof
- Design a TM $M_G$ that has $G$ built into it and use the result of $A_{CFG}$.
- $M_G =$ “On input $w$
  1. Run TM $S$ (from Theorem 4.7) on input $\langle G, w \rangle$
  2. If $S$ accepts, accept, otherwise reject."
THEOREM 4.11

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

- Note that \( A_{TM} \) is Turing-recognizable. Thus this theorem when proved, shows that recognizers are more powerful than deciders.
- We can encode TMs with strings just like we did for DFA’s (How?)
Acceptance Problem for TMs

- The TM $U$ recognizes $A_{TM}$
- $U = “On input \langle M, w \rangle where M is a TM and w is a string:
  1. Simulate $M$ on $w$
  2. If $M$ ever enters its accepts state, accept; if $M$ ever enters its reject state, reject.
- Note that if $M$ loops on $w$, then $U$ loops on $\langle M, w \rangle$, which is why it is NOT a decider!
- $U$ can not detect that $M$ halts on $w$.
- $A_{TM}$ is also known as the Halting Problem
- $U$ is known as the Universal Turing Machine because it can simulate every TM (including itself!)
Let $A$ and $B$ be any two sets (not necessarily finite) and $f$ be a function from $A$ to $B$.

- $f$ is **one-to-one** if $f(a) \neq f(b)$ whenever $a \neq b$.
- $f$ is **onto** if for every $b \in B$ there is an $a \in A$ such that $f(a) = b$.

We say $A$ and $B$ are the **same size** if there is a one-to-one and onto function $f : A \rightarrow B$.

Such a function is called a **correspondence** for pairing $A$ and $B$.

- Every element of $A$ maps to a unique element of $B$
- Each element of $B$ has a unique element of $A$ mapping to it.
Let \( \mathcal{N} \) be the set of natural numbers \( \{1, 2, \ldots\} \) and let \( \mathcal{E} \) be the set of even numbers \( \{2, 4, \ldots\} \).

\[ f(n) = 2n \] is a correspondence between \( \mathcal{N} \) and \( \mathcal{E} \).

Hence, \( \mathcal{N} \) and \( \mathcal{E} \) have the same size (though \( \mathcal{E} \subset \mathcal{N} \)).

A set \( A \) is \textit{countable} if it is either finite or has the same size as \( \mathcal{N} \).

\[ Q = \{ \frac{m}{n} \mid m, n \in \mathcal{N} \} \] is countable!

\( \mathbb{Z} \) the set of integers is countable:

\[ f(n) = \begin{cases} \frac{n}{2} & n \text{ even} \\ -\frac{n+1}{2} & n \text{ odd} \end{cases} \]
**The Diagonalization Method**

**Theorem**

\( \mathcal{R} \) is uncountable

**Proof.**

- Assume \( f \) exists and every number in \( \mathcal{R} \) is listed.

- Assume \( x \in \mathcal{R} \) is a real number such that \( x \) differs from the \( j^{th} \) number in the \( j^{th} \) decimal digit.

- If \( x \) is listed at some position \( k \), then it differs from itself at \( k^{th} \) position; otherwise the premise does not hold.

- \( f \) does not exist

---

\[
\begin{array}{c|c}
 n & f(n) \\
\hline
 1 & 3.14159\ldots \\
 2 & 55.77777\ldots \\
 3 & 0.12345\ldots \\
 4 & 0.50000\ldots \\
 \vdots & \vdots \\
 x & .4527\ldots \\
\end{array}
\]

defined as such, can not be on this list.
Some languages are not Turing-recognizable.

Proof

- For any alphabet $\Sigma$, $\Sigma^*$ is countable. Order strings in $\Sigma^*$ by length and then alphanumerically, so $\Sigma^* = \{s_1, s_2, \ldots, s_i, \ldots\}$

- The set of all TMs is a countable language.
  - Each TM $M$ corresponds to a string $\langle M \rangle$.
  - Generate a list of strings and remove any strings that do not represent a TM to get a list of TMs.
The set of infinite binary sequences, \( B \), is uncountable. (Exactly the same proof we gave for uncountability of \( \mathcal{R} \))

Let \( \mathcal{L} \) be the set of all languages over \( \Sigma \).

For each language \( A \in \mathcal{L} \) there is unique infinite binary sequence \( \chi_A \)

- The \( i^{th} \) bit in \( \chi_A \) is 1 if \( s_i \in A \), 0 otherwise.

\[
\Sigma^* = \{ \epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots \}
\]

\[
A = \{ 0, 00, 01, 000, 001, \ldots \}
\]

\[
\chi_A = \{ 0 \ 1 \ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ \ldots \}
\]
The function $f : \mathcal{L} \rightarrow \mathcal{B}$ is a correspondence. Thus $\mathcal{L}$ is uncountable.

So, there are languages that cannot be recognized by some TM.

There are not enough TMs to go around.
The Halting Problem is Undecidable

**Theorem**

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

**Proof**

- We assume \( A_{TM} \) is decidable and obtain a contradiction.
- Suppose \( H \) decides \( A_{TM} \)

\[
H(\langle M, w \rangle) = \begin{cases} 
{\text{accept}} & \text{if } M \text{ accepts } w \\
{\text{reject}} & \text{if } M \text{ does not accept } w 
\end{cases}
\]
We now construct a new TM $D$

$D = \text{"On input } \langle M \rangle, \text{ where } M \text{ is a TM}

1. Run $H$ on input $\langle M, \langle M \rangle \rangle$.
2. If $H$ accepts, reject, if $H$ rejects, accept”

So

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

When $D$ runs on itself we get

$$D(\langle D \rangle) = \begin{cases} 
\text{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\
\text{reject} & \text{if } D \text{ accepts } \langle D \rangle
\end{cases}$$

Neither $D$ nor $H$ can exist.
What Happened to Diagonalization?

Consider the behaviour of all possible deciders:

<table>
<thead>
<tr>
<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>$\ldots$</th>
<th>$\langle M_j \rangle$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>$\ldots$</td>
<td>accept</td>
</tr>
<tr>
<td>$M_2$</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>$\ldots$</td>
<td>accept</td>
</tr>
<tr>
<td>$M_3$</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>$\ldots$</td>
<td>reject</td>
</tr>
<tr>
<td>$M_4$</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>$\ldots$</td>
<td>accept</td>
</tr>
<tr>
<td></td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td></td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$D = M_j$</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>$\ldots$</td>
<td>$?$</td>
</tr>
<tr>
<td></td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td></td>
<td>$\vdots$</td>
</tr>
</tbody>
</table>

$D$ computes the opposite of the diagonal entries!
A language is **co-Turing-recognizable** if it is the complement of a Turing-recognizable language.

A language is decidable if it is Turing-recognizable and co-Turing-recognizable.

\(A_{TM}\) is not Turing recognizable.

- We know \(A_{TM}\) is Turing-recognizable.
- If \(A_{TM}\) were also Turing-recognizable, \(A_{TM}\) would have to be decidable.
- We know \(A_{TM}\) is not decidable.
- \(A_{TM}\) must not be Turing-recognizable.