Announcements
1. Homework 8 is available online.

Recursive, Recursively Enumerable, not even R.E.
1. Today, we will continue to explore how to categorize the computability of various languages into three categories:

DECIDABLE/RECURSIVE
RECURSIVELY ENUMERABLE
NOT RECURSIVELY ENUMERABLE

A\_TM
DFA
E\_TM

Turing-decidable/recursive We say a language \( L \) is Turing-decidable if there exists a TM \( M \) that halts on all inputs for which \( L = L(M) \).

Turing-recognizable/recursively enumerable We say a language \( L \) is Turing-recognizable if there exists a TM \( M \) for which \( L = L(M) \).

Not Turing-recognizable Eventually, we will further distinguish languages that are not recognizable but whose complements are recognizable from those where both the language itself and its complement are not recognizable.

Why Recursively Enumerable is Always Recognizable
1. Last time, we concluded that it was clear that any recursively enumerable language is also recognizable, because you can just build a TM that uses the enumerator for the language as a subroutine and searches for a match with its own input.
2. Now, we want to show that any recognizable language is also recursively enumerable.
3. The basic idea is that given a machine \( R \) that recognizes some language \( L \), we can build a machine \( E \) that uses \( R \) to check every string over its alphabet to see if \( R \) accepts and writes all the accepted strings on its tape.
4. We have to be very careful because \( R \) may loop on any \( w \in L \). If we just simulate \( R \) on every element of \( w_0, w_1, w_2, \ldots \) in order our simulator may get stuck in a loop on some early member of the sequence.
5. We solve this using a technique called dovetailing. We will design a simulator that simulates \( R \) processing many strings at a time. At each round, our simulator will simulate one step of \( R \) on each string it is currently simulating and then add one more string to the mix.
6. Our machine \( E \) will have three tapes:
   - One will hold the latest string in an enumeration of all strings over \( L \)'s input alphabet.
   - One will hold a sequence of strings representing triples corresponding to configurations reachable by \( R \) on certain inputs together with the input on which the computation that led to the configuration began. That is, each item on the tape might look like \( (u, q, v)\#w \) where \( (u, w, v) \) is a configuration that \( R \) could reach during a computation that started with \( w \) as input. This sequence of configurations will be divided by special markers into a prefix of configurations that have already been expanded, a middle section of configurations that are currently being expanded, and a suffix that still need to be expanded.
The last tape will hold the sequence of strings in $L$.

7. The machine will execute the following algorithm:
   - Initialize the first tape with $\epsilon$.
   - Initialize the second tape with $(\epsilon, q_0, \epsilon)\#\epsilon$.
   - Repeatedly (forever):
     - Place a marker at the end of the tape to separate the configurations that will be expanded in this iteration from those added in this iteration.
     - For each unexpanded configuration before this marker:
       * Write the next configuration it would yield at the end of the input tape.
       * Move the marker past this configuration to indicate that it has been expanded.
       * If the new configuration is in the accept state, write the input string that started this computation on the output tape.
     - Remove the marker that was used to mark the end of the sequence of configurations that were begin expanded on this iteration.
     - Replace the string $w$ on the first tape with $w'$, the next string over $M$’s alphabet.
     - Add a configuration $(\epsilon, q_0, w')\#w'$ to the end of the second tape.

**Closure Properties**

1. A final exercise that might cement our understanding of the differences between decidable, recognizable, and non-recognizable languages is to consider their closure properties.
   - If $A$ and $B$ are decidable languages with deciders $M_A$ and $M_B$, then
     - We can decide $AB$ using a non-deterministic machine that nondeterministically guesses where to divide its input up into an $A$ prefix and a $B$ suffix and then simulates $M_A$ and $M_B$ on the substrings to verify its guess.
     - We can decide $\overline{A}$ by just interchanging the accept and reject states of $M_A$.
   - The same simulations/arguments work for union, intersection and concatenation if $A$ and $B$ are Turing-recognizable, but not for complement.
   - If both $A$ and $\overline{A}$ are Turing-recognizable, then $A$ must be decidable.
     - Given TMs that recognize $A$ and $\overline{A}$ we could run them in parallel on any input on a 2-tape TM. If the $A$ machine accepted we would accept. If the $\overline{A}$ machine accepted, we would reject. If both sets were recognizable, one of the two would happen eventually, so the combined machine would decide the language $A$.
   - As a result, if there are any languages that are recognizable but not decidable, then recognizable languages must not be closed under complement. In fact, in that case, there must be some recognizable language whose complement is not recognizable.

**A Recognizable, but Undecidable Language**

1. Recall the language
   
   $$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } w \in L(M) \}$$
   
   We know that this language is recognizable. We believe and would like to prove it is undecidable.

2. To make it more obvious that the representation used for TMs will not matter, let’s restrict ourselves to binary TMs (i.e., TMs with input alphabet $\Sigma = \{0, 1\}$) and consider:
   
   $$A_{BTM} = \{ \langle M \rangle w \mid \langle M \rangle \text{ is a binary encoding of a binary TM, } \& \ w \in L(M). \} \subset \{0, 1\}^*$$
I believe that this change of language will make things a bit clearer because if we limit our attention to both binary TMs and binary encodings of TMs, the “w” in the input to $A_{BTM}$ will not need to be encoded in any way (as it is in the language $A_{TM}$ to enable us to include TMs over any input language we can think of).

It should be obvious that we can encode a TM in binary. All we need to do is encode a 5-tuple for each element of the transition function $\delta(q, x) = (q', y, L/R)$. We can encode both state number $n$ and tape alphabet number $n$ as $10^n$ and use double 1s to separate components of the tuple. A triple 111 can then be used to separate the description of the TM from its input $w$.

**Theorem:** $A_{BTM}$ is undecidable.

**Proof:** Suppose that $A_{BTM}$ was decidable. Then there would exist some TM $N$ that always halted such that $A_{BTM} = L(N)$.

- Given $N$, we could construct another TM $D$ which on any input $w$, made a copy of $w$ after its original input to form $ww$ and then ran $N$ on the result. This machine would decide the language

$$L(D) = \{ \langle M \rangle | \langle M \rangle \text{ is an encoding of a binary TM}$$

$$& \& \langle M \rangle \in L(M). \}$$

- Now, suppose that we alter $D$ just a bit to produce a new machine named $\overline{D}$. $\overline{D}$ will be identical to $D$ except its accept and reject states will be interchanged. Since all of these machines are deciders, we can say

$$L(\overline{D}) = \{ \langle M \rangle | \langle M \rangle \text{ is not an encoding of a binary TM}$$

$$& \& \langle M \rangle \notin L(M). \}$$

- Now, consider what happens when we apply $\overline{D}$ to its own description. That is, we apply $\overline{D}$ to the input $\langle \overline{D} \rangle$. Since $\langle \overline{D} \rangle$ is clearly an encoding of a binary TM, we can see that $\langle \overline{D} \rangle \in L(\overline{D}) \equiv \langle \overline{D} \rangle \notin L(\overline{D})$.

This is nonsense! Or better yet a contradiction. As a result, we can state that our original assumption that $A_{BTM}$ was decidable must be false.

**What Diagonal?**

1. The proof that $A_{BTM}$ is undecidable is described as a diagonalization proof.

2. You may (or may not!) recall that on the first day of class we used a diagonalization argument to show that there were more reals than integers.

- We assumed that there was a mapping from the natural numbers to the reals. That is, that there was some list that included every real number in such a way that we could identify some real as number 1, some real as number 2, and so on.

- We then described a real number that could not be in this list by stipulating that we would select the $i$th digit of the decimal expansion of our number to be different from the $i$th bit of the $i$th number in our list of reals.

- Since a real number constructed in this way could not be in our list, this contradicted the assumption that such a list could exist.

- To see the diagonal in this argument, consider the following table which is supposed to show the first few digits of each of the first few real numbers in some potential numbered list of all real numbers.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>.</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>.</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>.</td>
<td>9</td>
<td>9</td>
<td>7</td>
<td>9</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>.</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>.</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>.</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>I</td>
<td>$d_{8,1}$</td>
<td>$d_{8,2}$</td>
<td>$d_{8,3}$</td>
<td>$d_{8,4}$</td>
<td>$d_{8,5}$</td>
<td>$d_{8,6}$</td>
<td>$d_{8,7}$</td>
</tr>
</tbody>
</table>
The \( i \) digit of the number we construct to show that such a table cannot contain every real number is chosen to be different from \( d_{i,i} \), the \( i \)th digit of the \( i \)th number. These are the numbers found along the diagonal of this table (if you ignore any portion of one of the real numbers shown before the decimal point).

- Note that it is not necessary for each real to appear just once in the purported enumeration of all reals to make this work. Even if some one real number appeared infinitely often, the argument would still work as long as we assumed every real appeared at least once.

3. The machine \( D \) we constructed in our proof that \( A_{BTM} \) is undecidable can be viewed as diagonalizing over a list of all TMs in the same way we formed a real number by diagonalizing over a purported list of all reals.

4. To see this, we first have to recognize that we can see the set of binary strings as a sequence of numbers.

- This may seem obvious at first. Just count in binary! Right?
  
  \[
  0, 1, 10, 11, 100, 101, \ldots
  \]

- Unfortunately, this only works if you leave out some troublesome members of \( \{0,1\}^* \):

  \[
  00, 01, 010, 0010, 0000000001, \text{etc.}
  \]

Every number corresponds to infinitely many members of \( \{0,1\}^* \) because leading 0s don’t change the interpretation of a string as a number.

- There is a quick fix. Given \( w \in \{0,1\}^* \), we can number \( w \) uniquely using the value obtained if we interpret \( 1w \) as a binary number.

\[
\begin{align*}
\epsilon & \Rightarrow 1 &= w_1 \\
0 & \Rightarrow 10 &= w_2 \\
1 & \Rightarrow 11 &= w_3 \\
00 & \Rightarrow 100 &= w_4 \\
01 & \Rightarrow 101 &= w_5 \\
10 & \Rightarrow 110 &= w_6 \\
11 & \Rightarrow 111 &= w_7 \\
000 & \Rightarrow 1000 &= w_8
\end{align*}
\]

5. Using this numbering of binary strings, we can associate each natural number with a not-necessarily unique binary TM using the encoding for binary TMs chosen for our definition of \( A_{BTM} \).

- The “not-necessarily unique” caveat covers two issues.
  - There are many encoding of any valid binary TM in just about any scheme we might come up with since just reordering the touples that describe the transition function.
  - Some binary strings are not valid encodings at all. For the purpose of associating each string with a TM we will consider all such strings to describe a TM that accepts no input strings.

- Thus, we will say that \( M_n \) is the TM described by the \( n \)th string over \( \{0,1\} \), \( w_n \) if \( w_n \) is a valid TM encoding and that \( M_n \) is a TM that accepts nothing otherwise.

6. We can now imagine an infinite table whose rows correspond to TMs in the ordering this numbering induces and whose columns correspond to the ordered sequence of binary strings.

\[
\begin{array}{cccccccc}
& w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & w_7 & \ldots \\
M_1 & \text{reject} & \text{reject} & \text{reject} & \text{reject} & \text{reject} & \text{reject} & \text{reject} & \text{...} \\
M_2 & \text{reject} & \text{reject} & \text{reject} & \text{reject} & \text{reject} & \text{reject} & \text{reject} & \text{...} \\
M_3 & \text{accept} & \text{reject} & \text{accept} & \text{reject} & \text{reject} & \text{accept} & \text{reject} & \text{...} \\
M_4 & \text{accept} & \text{accept} & \text{accept} & \text{accept} & \text{accept} & \text{accept} & \text{accept} & \text{...} \\
M_5 & \text{reject} & \text{reject} & \text{reject} & \text{reject} & \text{reject} & \text{accept} & \text{accept} & \text{...} \\
M_6 & \text{reject} & \text{accept} & \text{reject} & \text{accept} & \text{reject} & \text{reject} & \text{...} \\
M_7 & \text{accept} & \text{accept} & \text{reject} & \text{reject} & \text{accept} & \text{reject} & \text{...} \\
M_8 & \text{reject} & \text{accept} & \text{accept} & \text{accept} & \text{accept} & \text{reject} & \text{...} \\
\end{array}
\]

Each cell indicates whether the TM for the row accepts the input for that column.

7. The machine \( D \) we described in our proof that \( A_{TM} \) is undecidable corresponds to the list of accept/reject results listed along the diagonal of this table.

8. If the machine \( \overline{D} \) we described in our proof by contradiction could exist, its language \( L(\overline{D}) \) would be described by the opposite of the sequence of “reject”s and “accept”s found along the diagonal of this
table. Thus, it will be different from every row of the table in at least one position.

9. Unlike the diagonalization proof that Cantor used to show that the reals were not countable, the construction of the machine $D$ does not lead to the conclusion that the set of TMs cannot be enumerated. We know that TMs can be enumerated! Instead, it leads to the conclusion that $D$ cannot be part of the list of all TMs. That is, $D$ does not exist.

**Reduction**

1. With regular languages and context-free languages, the appropriate pumping lemma was used over and over again to show that languages did not belong to the class in question.

2. The diagonalization technique is not used repeatedly like the pumping lemmas. Instead, we use the fact that $A_{TM}$ is now known to be recognizable but undecidable together with closure properties and reductions to show almost all other similar results.

3. As a first example, we can show that it is not necessary to restrict our attention to machines with binary input alphabets encoded in binary. Instead, we can consider the broader language.

$$A_{TM} = \{ (M, w) \mid M \text{ is a TM and } w \in \mathcal{L}(M) \}$$

- Suppose that $A_{TM}$ were decidable. In this case, there would be some machine $M$ that decided $A_{TM}$.
- The definition of this language would include some scheme for encoding TM descriptions that could handle machines and inputs over any input alphabet. Suppose we constructed a machine $M'$ with a binary input alphabet that first rejected its input if it was not a valid binary encoding of a binary TM and its input, $(B)w$. Otherwise, it would write the translation of this input in the encoding used by $M$ on its tape and then run $M$.
- Since we assumed $M$ decided $A_{TM}$, $M'$ would have to decide $A_{BTM}$. We just proved, however, that it is impossible to decide $A_{BTM}$. Based on this contradiction, we can conclude that no decider for $A_{TM}$ exists. That is, we have shown that $A_{TM}$ is undecidable.

**Languages that are not Recognizable**

1. In previous lectures, we showed that several languages (mainly involving properties of regular or context-free languages) were decidable.

2. Using diagonalization and reduction, we have now shown that $A_{BTM}$ and $A_{TM}$ are not decidable, but both of these languages are recognizable.

3. The last bubble in out Venn diagram that we need to fill with an example is the area for languages that are not even recognizable. This turns out to be easy.

4. Earlier I pointed out that if both a language and its complement are recognizable then they must both also be decidable.

   - If both $A$ and $\overline{A}$ are recognizable, they must be recognized by some pair of machines $M$ and $\overline{M}$.
   - We can build a machine that simulates both of these machines in parallel on the same input (it should be easy to see how to do this on a two-tape TM).
   - We can then decide $A$ by accepting if the simulation of $M$ accepts and rejecting if the simulation of $\overline{M}$ accepts. One of the two must happen eventually.

5. As a result, now that we know that $A_{BTM}$ and $A_{TM}$ are not decidable we can immediately conclude that $\overline{A}_{BTM}$ and $\overline{A}_{TM}$ are not recognizable.