Announcements

1. Homework 8 due Friday

Why Recursively Enumerable is Equivalent to Recognizable

1. We say that a language is recursively enumerable if we can build a TM, $E$, that will write a sequence of strings that belong to $L$ on one of its tapes in such a way that every $w \in L$ will eventually appear in this sequence. As a result, we can think of the machine as numbering the elements of $L$ (it would be easy though time consuming to eliminate any duplicates). Each $w \in L$ is associated with the number of the position at which it appears within the sequence output by $E$.

2. Last class, we saw that every recursively enumerable language $L$ is recognizable.

   • Given an enumerator $E$ for a language, we can build a recognizer $R$ for the same language by having $R$ run $E$ as a sub-machine and every time $E$ writes a new member of $L$ on its tape compare that member to $R$’s input. If they match, $R$ accepts.

3. Slightly more surprising (and subtle to prove) is the fact that every Turing-recognizable language is also recursively enumerable.

4. 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.

5. We have to be very careful because $R$ may loop on any $w_i \notin L$. 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.

6. 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.

7. 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 mid-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 $L$.

8. 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.

Click here to view the slides for this class
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 \( A \cup B \) or \( A \cap B \) by using a two-tape TM to simulate \( M_A \) and \( M_B \) simultaneously and then appropriately combine their decisions.
     - 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. It is important to realize that it is a bit hard to do union with a deterministic TM. To accomplish this the machine has to interleave the simulation of machines for the individual languages. An easier argument is to have a non-deterministic machine guess which of the languages in the union to check.
   - The complement of a recognizable language is not necessarily recognizable. It should be clear that \( \overline{E_{TM}} \) is a recognizable language, but its complement \( E_{TM} \) is a language that seems hard to recognize (we will prove it is impossible shortly).
   - 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 (we haven’t proved such a language exists yet), 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
   
   \[ A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } w \in \mathcal{L}(M) \} \]

   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:
   
   \[ A_{BTM} = \{ \langle M \rangle w \mid \langle M \rangle \text{ is a binary encoding of a binary TM, } \& \ w \in \mathcal{L}(M). \} \subset \{0,1\}^* \]

   - 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
  \[
  L(D) = \{ \langle M \rangle \mid \langle M \rangle \text{ is an encoding of a binary TM and } \langle M \rangle \in L(M). \}
  \]

- 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
  \[
  L(\overline{D}) = \{ \langle M \rangle \mid \langle M \rangle \text{ is not an encoding of a binary TM or } \langle M \rangle \notin L(M). \}
  \]

- 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.