Announcements
1. Homework 7, problems 1, 2 and 5 due today.
2. Homework 8 available this weekend.
3. Midterms will be graded this year!

Decidable Properties of DFAs/Reg Exps., CFGs
1. Last time we began looking at a collection of decidable languages that are based on questions about formalisms for describing languages.
   - \( A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA and } w \in \mathcal{L}(B) \} \)
   - \( A_{REX} = \{ \langle R, w \rangle \mid R \text{ is a regular expression and } w \in \mathcal{L}(R) \} \)
   - \( E_{DFA} = \{ \langle A \rangle \mid A \text{ is a DFA and } \mathcal{L}(A) = \emptyset \} \)
   - \( A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG and } w \in \mathcal{L}(G) \} \)
   - \( E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } \mathcal{L}(G) = \emptyset \} \)

2. Embarrassingly, I am a bit unsure where we left off! I think we talked about \( E_{DFA} \) and \( EQ_{DFA} \) but a) I am not confident and b) I bet a bit of review won't hurt anyway so...

3. Let’s think a bit about how we could argue that the following two languages are decidable.
   - \( E_{DFA} = \{ \langle A \rangle \mid A \text{ is a DFA and } \mathcal{L}(A) = \emptyset \} \), and
   - \( EQ_{DFA} = \{ \langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } \mathcal{L}(A) = \mathcal{L}(B) \} \)

4. Both of these can be shown using results we showed (much) earlier.

5. For \( E_{DFA} \):
   - We could then simulate the DFA on each of these strings and reject if we find a string the DFA accepts.
   - The problem is, we don’t want to wait until we have tried every string before rejecting. If we did this, our machine would not halt!
   - The pumping lemma allows us to conclude that if the is any string in the language of the DFA, that string must be less than the pumping length. This is because if the shortest string in the language was longer than the pumping length, pumping down would produce a shorter string which therefore could not be in the language as the lemma requires.
   - The number of states in the DFA is an upper bound on the pumping length, so we can reject a DFA description input to our machine one we have checked that no strings shorter than the number of states plus 1 are accepted by the DFA.

6. Alternately, we can treat \( E_{DFA} \) as a graph search problem. Viewing the machine’s state diagram as a graph, if there is any path from the start symbol to a final state in the graph, then the machine must accept the string whose symbols label the edges of the path. A breadth-first search of the graph will find all reachable states.

7. For \( EQ_{DFA} \), recognize that \( A = B \equiv (A - B) \cup (B - A) = (A \cap \overline{B}) \cup (B \cap \overline{A}) = \emptyset \). We know that given DFAs that recognize \( A \) and \( B \) we can construct a DFA to recognize \( (A \cap \overline{B}) \cup (B \cap \overline{A}) \). Therefore, we can decide whether \( A = B \) by deciding whether the DFA we construct to recognize \( (A \cap \overline{B}) \cup (B \cap \overline{A}) \) accepts the empty language.

8. For \( A_{CFG} \), we can easily imagine a nondeterministic TM that would recognize the language. It would just guess a derivation using the grammar’s productions and verify that it yields \( w \). Unfortunately, if a grammar has rules like \( P \rightarrow Q \) and \( Q \rightarrow P \), there can be derivations of unbounded length, so the nondeterministic machine would be a recognizer for this language, but not a decider. Therefore, we need some way to put a bound on the length of the derivations we need to consider.

9. In class, we saw an algorithm to convert a CFG into Chomsky normal form. If we build a TM that converts its input CFG into Chomsky
normal form, it can then limit its search to derivations of length at
most \(2w + 1\). This makes our machine a decider.

10. \(E_{CFG}\) is a bit trickier. The decidability of this language depends on
an algorithm that determines for each variable or terminal in a CGF
whether that symbol derives any string composed entirely of terminals.
Given a CFG \(G = (V, \Sigma, R, S)\), we iteratively compute a set \(USEFUL\)
of symbols known to derive some string of terminals as follows:

- Initially, set \(USEFUL = \Sigma\)
- Repeatedly (until the following process does not increase the size
  of \(USEFUL\))
  - For each \(V \rightarrow \beta \in R\) where \(V \notin USEFUL\),
    - if every \(s \in (\Sigma \cup V)\) in \(\beta\) is already in \(USEFUL\), then
      add \(V\) to \(USEFUL\)

If at the end of this algorithm, \(S \in USEFUL\), then we know that
\(L(G) \neq \emptyset\), otherwise, \(L(G) = \emptyset\).

Recursive, Recognizable, not even Recognizable, and worse!

1. We have not considered one of the languages in our original list:
   - \(EQ_{CFG} = \{\langle G, H \rangle \mid G\) and \(H\) are CFGs and \(L(A) = L(B)\}\)

2. In addition, we have not considered the decidability of similar lan-
guages about TMs:
   - \(A_{TM} = \{\langle M, w \rangle \mid M\) is a TM and \(w \in \mathcal{L}(M)\}\)
   - \(E_{TM} = \{\langle M \rangle \mid M\) is a TM and \(\mathcal{L}(M) = \emptyset\}\)
   - \(EQ_{TM} = \{\langle M, N \rangle \mid M\) and \(N\) are TMs and \(\mathcal{L}(M) = \mathcal{L}(N)\}\)

3. In case you cannot guess, these are all examples of languages that are
not decidable. In fact, one of them is not even Turing-recognizable.

4. Before proving these facts, I want to take some time to make sure we
are as comfortable as we can get with the terminology we will be using
to distinguish different degrees of computability relative to the Turing
machine model.

5. We will focus on three types of languages. The diagram below sum-
marizes the inclusion relationships that exist between these three sets
of languages and several others we have (or will) encounter.

Turing-decidable We say a language \(L\) is Turing-decidable if there
exists a TM \(M\) that halts on all inputs for which \(L = L(M)\). The
terms “decidable” and “recursive” are synonymous with “Turing-
decidable”. All of the languages we discussed so far this semester
including regular and context-free language belong to this group.

Turing-recognizable We say a language \(L\) is Turing-recognizable if
there exists a TM \(M\) for which \(L = L(M)\). Basically, we have sim-
ply dropped the requirement that \(M\) halt on inputs that are not
in \(L\). Therefore, all Turing-decidable languages are also Turing-
recognizable. The terms “recognizable” and “recursively enumer-
able” are synonymous with “Turing-recognizable”.

- It should be clear that
  \[ A_{TM} = \{\langle M, w \rangle \mid M\) is a TM and \(w \in \mathcal{L}(M)\}. \]
is Turing-recognizable. Just as a TM can simulate the computation of a DFA, we can design a TM to simulate any other TM as long as it is given a description of that TM and the input that machine should process. If \( w \in L(M) \), then the simulation will eventually reach an accepting state and the simulator can accept \( <M, w> \). However, it is not clear that this language is decidable since if \( M \) does not halt on \( w \), there is no obvious way that the simulator can distinguish this from a very long computation that will eventually accept so it will not halt either.

Note: This is not intended as a proof that \( A_{TM} \) is not decidable. The goal is just to help us clearly understand the structure of these classes of language before we do prove that various language fall into various categories.

**Not Turing-recognizable** Each Turing-recognizable language is associated with one or more (actually always more) Turing machines. We know that there are only countably many TMs. Therefore, there can only be countably many Turing-recognizable languages. Since there are uncountably many subsets of the set of all strings over any alphabet, there must be many languages that are not Turing-recognizable (a.k.a. not recursively enumerable).

Intuitively, it should seem likely that

\[
E_{TM} = \{(M) \mid M \text{ is a TM and } L(M) = \emptyset\}
\]

is such a language. The only obvious way to determine that the language of a TM is empty is to check every string to make sure that there is no string that the machine accepts. Since there are infinitely many strings, this process will not terminate if the language is actually empty.

The diagram below summarizes the language we have suggested might belong to each of these three sets of languages.

---

**Enumerability**

1. It is probably worth a little (or maybe not so little) historical note to explain the name “recursively enumerable”.

2. 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 \( E \).

3. It should be clear 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.

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