Announcements
1. Homework 2 now available.

Review
1. Last time, we used our formal definition of deterministic finite automata do:
   - Allow us to define and verify the correctness of families of languages. (We did \( L_{\text{DivN}} = \) the set of binary strings representing numbers divisible by \( N \).)
   - Allow us to develop theorems we can use as tools to show that languages are regular without describing appropriate DFAs. (We observed that the set of regular languages is closed under complement and intersection.)

2. To polish off the topic of closure for regular languages under simple set operations, consider the union of two regular sets.
   - One can prove that regular languages are closed under union using a construction similar to what we saw last class:

   **Proof:** Suppose that we know that two languages \( L_1 \) and \( L_1 \) over the same alphabet \( \Sigma \) are regular. Then we know that there must be two DFAs \( M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1) \) and \( M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2) \) such that \( L_1 = L(M_1) \) and \( L_2 = L(M_2) \).
   Consider the machine
   \[
   M = (Q_1 \times Q_2, \Sigma, \delta, (s_1, s_2), F_1 \times Q_2 \cup Q_1 \times F_2)
   \]
   where \( \delta((q_1, q_2), x) = (\delta_1(q_1, x), \delta_2(q_2, x)) \).
   - Better yet, we can take a simpler approach by observing that \( A \cup B = \overline{A} \cap \overline{B} \).

A Little More Practice
1. As both review and motivation for our next topic, I would like you to break into groups and work on sketching a DFAs for the following language:
   (a) The language of binary strings ending in 110.

Nondeterministic Finite Automata
1. I am guessing that the solutions you found to the exercise above look something like:

2. Let’s consider an alternate approach to describing the first language.
   - If you look carefully at the state diagram for this DFA, you will notice that the “most important” path runs straight from the “pre” (for prefix) state to the final state. This path is shown in red below:
• Most of the non-red edges in this machine show what to do when something that looked like the final 110 turned out not to be!

• It would be nice if we could describe this language with a state diagram that wasn’t so cluttered with edges that said what to do in the non-interesting cases. It would be nice if the final machine looked more like:

• This clearly isn’t good enough since it only accepts the string 110. We somehow need to add transitions that allow the machine to stay in the “prefix” state until it is up to the last three (hopefully) matching symbols. To do this, we would have to add transitions like:

• This transition diagram is an example of a nondeterministic finite automaton (or NFA). It has two features we would not (strictly) allow in a DFA:
  – Some states (“pre”) have more than one outgoing edge labeled with the same symbol. Basically, the state diagram gives this machine a choice when it is in state “pre” and sees a 1. If it doesn’t think that the 1 it is looking at is part of the final 110, it can just take the looping edge and stay in state “pre”. On the other hand, if it is “feeling lucky” it can follow the edge to state 1 and if its is indeed lucky it will end up in the final state “110” at the end of its input.
  – Some states have no transition on some symbol. Actually, this is an accepted notational shorthand when drawing DFAs. If a transition is omitted it means that symbol should take you to a non-accepting state from which there is no escape. For NFAs, however, we will include this as a fundamental part of the definition rather than a notational convenience.

**Understanding NFAs**

1. There are several ways of interpreting NFAs.
   • The text suggests a model in which non-determinism is tied to parallelism. That is, when more than one path is possible, the machine effectively clones itself and follows all paths accepting if any of the simultaneous computations succeeds.
   We can illustrate this idea by marking all the states we might be in at each step of processing some input (like 11110110).
• I was brought up on (and prefer) a different interpretation which I tried to hint at above. In this version we assume that our machine is just a very good guesser. At each point where multiple transitions are possible, the machine guesses which one to take and if there is a sequence of guesses that will eventually reach a final state, the machine somehow magically guesses that alternative. This probably sounds silly, but if you accept it for now I will try to give some justification for this view later.

**More Closure Properties**

1. The idea of a machine that is really good at guessing should seem a bit silly. So it may help to see that making this idea make sense may have more significance than letting us recognize strings that end in or contain 110 easily.

2. One way that we can make use of NFAs is to devise simple proofs of some more closure properties.

   - When we introduced the notion of a language as a set of strings, we also introduced a product operation specific to sets of strings in which the strings in the product were formed by concatenating two strings from the languages to which the product is applied.

   - Consider how we might show that regular languages are closed under this operation.

   - Given two regular languages $L_1$ and $L_2$, we know that there must be two DFAs $M_1$ and $M_2$ with $L_1 = L(M_1)$ and $L_2 = L(M_2)$. To show that $L_1L_2$ was regular, we would try to describe a way to combine the parts of $M_1$ and $M_2$ to form a new machine $M_3$ that recognized the product of the original languages.

   - The diagram below suggests the idea that $M_3$ would somehow have copies of $M_1$ and $M_2$ inside:

     - Any string in $L_1L_2$ must consist of a string from $L_1$ followed by a string from $L_2$. Therefore, if we start in the start state of $M_1$ and somehow once we get to a final state jump to the start state of $M_2$, each sub-machine can check that its part of the input string belongs to the right language. To accomplish this, we might make the final state of $M_1$ (we are assuming there is just one to keep this intuitive argument simple) and the start state of $M_2$ be the same state in $M_3$ as suggested below:

     - A machine produced in this way will not be a valid DFA. The problem is that both the final state of $M_1$ and the start state of $M_2$ will have their own outgoing transition arrows for each symbol
in the alphabet (such as “1”). As a result, the merged state will have two choices for its next state on many input symbols. This requires non-determinism.

**Formalizing NFA**

1. Just as we gave formal definitions to explain how to understand DFAs, we can do the same for NFAs.
   - Actually, what we will describe here is not quite the standard definition of NFAs. We are leaving out the possibility of "\( \epsilon \)-transitions. We will use the simplified version to understand some basic properties of NFAs and then extend our definition to the standard one later.

2. **Definition.** An NFA is a five tuple \( D = (Q, \Sigma, \delta, s, F) \) where:
   - \( Q \) is a finite set of states
   - \( \Sigma \) is the input alphabet
   - \( \delta : Q \times \Sigma \rightarrow \mathcal{P}(Q) \) is a state transition function
   - \( s \in Q \) is the start state
   - \( F \subseteq Q \) is a set of accept states

3. We define a function \( \hat{\delta} \), which extends \( \delta \) so that we can apply it to strings instead of just individual symbols:
   \[
   \hat{\delta}(\pi, \varepsilon) = \pi \quad (\pi \in \mathcal{P}(Q))
   \]
   \[
   \hat{\delta}(\pi, wx) = \bigcup_{q \in \delta(\pi, w)} \delta(q, x) \quad (\pi \in \mathcal{P}(Q), x \in \Sigma, w \in \Sigma^*)
   \]

4. **Definition** \( L(D) = \{w \mid w \in \Sigma^* \text{ and } \hat{\delta}(\{s\}, w) \cap F \neq \emptyset\} \)

   **Nondeterminism = Determinism?**

1. Let’s get right to the big result (almost) and prove that NFAs do not provide the ability to describe anything we could not have described with a DFA. Since every DFA is also a NFA, this means that the set of language recognized by NFAs is just the regular languages.
   - The “almost” refers to the fact that we really are not quite using the standard definition of NFAs yet. Working with the simplified definition will make the fundamental idea in the proof of equivalence stand out more clearly. Then, if we have time we will extend it to the standard definition of NFA.

2. So what we want to prove is

   **Theorem:** if \( L = L(N) \) for some nondeterministic finite automaton \( N \) then there is a deterministic finite automaton, \( D \) such that \( L(D) = L \).

3. Recall that when we showed that the intersection of two regular languages was regular we built a DFA each of whose states kept track of a pair of states, one from each of the machines that recognized the language whose intersection we were trying to recognize.

4. If a single state of one machine can represent two states of other machines, we could imagine making states in one machine that represented 3, 4, 5 ... states of one or more other machines. In particular, we can make one machine have states that represent the subsets of states of another machine.

5. With this in mind, if the machine \( N \) described in the statement of our theorem is \( N = (Q, \Sigma, \delta_N, s_N, F_N) \), we will build a machine \( D \) whose set of states has one state corresponding to each subset of \( Q \). That is, the states of \( D \) will be the set of all subsets of \( Q \) which is just the power set of \( Q \).

   \[
   D = (\mathcal{P}(Q), \Sigma, \delta_D, s_D, F_D)
   \]

6. We want our deterministic machine to use its states to keep track of all of the states \( N \) might be in after processing some \( w \). If \( w \) is \( \varepsilon \), \( N \) must be in \( s_N \). So, \( s_D \) should be \( \{s_N\} \).

7. Using similar reasoning we can see that \( F_D \) should contain all subsets of states that include any final state of \( N \). That is

   \[
   F_D = \{\pi \in \mathcal{P}(Q) \mid \pi \cap F_N \neq \emptyset\}
   \]
8. All that is left is to define $\delta_D$ so that it mimics the parallel model we discussed of how NFAs operate. That is, given a subset of states we might be in simultaneously we want $\delta_D$ to tell us the set of states we would be in after processing a particular input symbol. Therefore, as required:

$$\delta_D : \mathcal{P}(Q) \times \Sigma \rightarrow \mathcal{P}(Q)$$

9. In particular, we can define $\delta_D$ by using $\delta_N$ to figure out where we might go from each state of $N$ in the current state of $D$. That is:

$$\delta_D(\pi, x) = \bigcup_{q \in \pi} \delta_N(q, x)$$

10. CLEARLY, the $D$ we have described accepts the same language as $N$ (by “clearly” I mean that I don’t think we need to go through the exercise of proving in detail that $\delta_N(s_N, w) \cap F_N \neq \emptyset \iff \delta_D(\{s_N\}, w) \in F_D$).

11. As an example, consider how this construction would work when applied to the NFA shown below which recognizes strings that end in 10.

12. The resulting machine would look like the machine below.

The label on each set indicates the subset of states in the NFA that correspond to that state in the DFA.