Announcements

1. Homework 2 now available.

δ Blues

1. Last time I presented a definition from Sipser’s text that precisely explains how we can determine whether a DFA accepts a particular string.

Definition: We say that a FSA \( M = (Q, \Sigma, \delta, s, F) \) accepts a string \( w \) and write \( w \in L(M) \) if and only if for some sequences \( (w_1, \ldots, w_n) \) and \( (q_0, \ldots, q_n) \) where \( w_i \in \Sigma \) and \( q_i \in Q \):

- \( w = w_1 w_2 \ldots w_n \)
- \( q_0 = s \)
- \( q_i = \delta(q_{i-1}, w_i) \) for \( 1 \leq i \leq n \)
- \( q_n \in F \)

2. We will frequently use an alternate but equivalent approach that depends on a recursive definition of a function that extends \( \delta \).

- The transition function \( \delta \) tells us how a machine will respond to any single symbol of input. It is not, however, defined on strings (of length other than one). That is, while we may know that for some DFA \( \delta(x, a) = y \) and \( \delta(y, b) = z \), \( \delta(x, ab) \) is undefined.
- Let \( \hat{\delta} \) be a function from \( Q \times \Sigma^* \) to \( Q \) (instead of just \( Q \times \Sigma \) like \( \delta \)). That is, it operates on state/string pairs instead of just state/character pairs.
- We want \( \hat{\delta} \) to function as an extension of \( \delta \) that determines the final destination reached after performing all of the transitions associated with the symbols in its second parameter.

- Define \( \hat{\delta} \) inductively on the length of a string using \( \delta \):

\[
\hat{\delta}(q, \varepsilon) = q \quad (q \in Q) \\
\hat{\delta}(q, wx) = \delta(\hat{\delta}(q, w), x) \quad (q \in Q, x \in \Sigma, w \in \Sigma^*)
\]

- Note that \( \delta(q, a) \) in the definition refers to the transition table so \( \hat{\delta} \) and \( \delta \) agree on strings of length 1.

3. With \( \hat{\delta} \) in hand we can give a definition of acceptance:

Definition A string \( w \in \Sigma^* \) is accepted by a DFA \( D \) if and only if \( \hat{\delta}(s, w) \in F \).

4. The language of a DFA \( D \) follows naturally:

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

Closure Properties

1. Another advantage of having formal definitions for finite state machines and regular languages is that we can prove results showing that the set of regular language is closed under certain set operations.

- We say that a set is closed under an operation if performing the operation on elements of the set only produces other elements of the set. The integers, for example, are closed under addition but not under division.
- Since languages are just sets, it is interesting to ask whether the set of regular languages is closed under set operations like union, intersection and union.

2. Consider the DFA we showed earlier that recognizes the set of binary strings with even parity (i.e., with an even number of 1s):
Suppose we instead wanted to use odd parity. That is, we wanted a DFA that recognized the set of binary strings containing an odd number of 1s. What would this DFA look like?

- All we need to do is make the accepting state non-accepting and the non-accepting state accepting.

3. This is an example of a general property of regular languages. They are closed under complementation. That is, if $L$ regular language over $\Sigma^*$ then the language of all strings in $\Sigma^*$ not in $L$ is also regular.

Our formalism gives us a means to prove this:

**Theorem:** For any language $L$ over an alphabet $\Sigma^*$, the complement of $L$, $\bar{L}$ is also a regular language.

**Proof:** Given that $L$ is regular, we know that there exists some DFA $M = (Q, \Sigma, \delta, q_0, F)$ such that $L(M) = L$. To see that $\bar{L}$ is regular, we must show that there must also be some $M'$ such that $L(M') = \bar{L}$.

Let $M' = (Q, \Sigma, \delta, q_0, Q - F)$. Now we know that $w \in \bar{L} \iff w \notin L \iff w \notin L(M) \iff \hat{\delta}(q_0, w) \notin F \iff \delta(q_0, w) \in Q - F$. This shows that $\bar{L} = L(M')$ so we can conclude that $\bar{L}$ is regular.

4. Knowing that the set of regular languages is closed under some operation can provide the means to prove that a language is regular without constructing a DFA specifically to recognize the language. Consider how we could use a closure property to show that the language of binary strings in which both the number of 1s and the number of 0s was even.

- To keep things clear, let’s first rename the states of one of our favorite DFAs to indicate that they keep track of whether the number of 1s is even or odd:

- Now, we can easily describe a very similar machine that recognizes the set of strings containing an even number of 0s:

- The language we want to describe is the intersection of the languages recognized by these two machines. Therefore, if we can show that the intersection of any two regular languages (and are willing to assume the two machines above recognize the obvious
5. We can show that the union of two regular languages is regular using a very similar construction in which the set of final states is instead the all pairs of states that include at least one final state from either machine.

- 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

$$M = (Q_1 \times Q_2, \Sigma, \delta, (s_1, s_2), F_1 \times F_2)$$

where $\delta((q_1, q_2), x) = (\delta_1(q_1, x), \delta_2(q_2, x))$.
- The idea here is that each state $(q_1, q_2)$ of the new machine simultaneously keeps track of the state that $M_1$ would reach on the current input $(q_1)$ and the state that $M_2$ would reach on the current input $(q_2)$.
- We can show by induction that $\hat{\delta}((q_1, q_2), w) = (\hat{\delta}_1(q_1, w), \hat{\delta}_2(q_2, w))$

**Base Case:** $\hat{\delta}((q_1, q_2), \epsilon) = (q_1, q_2) = (\hat{\delta}_1(q_1, \epsilon), \hat{\delta}_2(q_2, \epsilon))$

**Inductive Case:** (Assume for $w$ then prove for $wx$.)

$$\hat{\delta}((q_1, q_2), wx) =$$

$$\delta(\hat{\delta}((q_1, q_2), w), x) = \quad \text{by def. of } \hat{\delta}$$

$$\delta(\hat{\delta}_1(q_1, w), \hat{\delta}_2(q_2, w)), x) = \quad \text{by ind. assumption}$$

$$(\hat{\delta}_1(q_1, w), \hat{\delta}_2(q_2, w)) \quad \text{by def. of } \hat{\delta}$$

$$(\hat{\delta}_1(q_1, w), \hat{\delta}_2(q_2, w)) \quad \text{by def. of } \hat{\delta}_1 \& \hat{\delta}_2$$

- From this we can see argue that $w \in L_1 \cap L_2 \iff w \in L_1$ and $w \in L_2 \iff \hat{\delta}_1(s_1, w) \in F_1$ and $\hat{\delta}_2(s_2, w) \in F_2 \iff (\hat{\delta}_1(s_1, w), \hat{\delta}_2(s_2, w)) \in F_1 \times F_2 \iff \hat{\delta}(s_1, s_2, w) \in F_1 \times F_2 \iff w \in L(M)$ as desired.

6. Better yet, we can take a simpler approach by observing that $A \cup B = \overline{A \cap B}$

### A Little More Practice

1. Take a moment to work on sketching a DFAs for the following language:
   - (a) The language of binary strings ending in 110.
2. I am guessing that the solutions you found to the exercise above look something like:

![DFA Diagram](image)

3. Let’s consider an alternate approach to describing the 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 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 in which 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 I will try to give some justification for this view later.

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 “$\varepsilon$-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
   $\delta : Q \times \Sigma \rightarrow P(Q)$ 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:

   $\hat{\delta}(\pi, \varepsilon) = \pi (\pi \in P(Q))$
   $\hat{\delta}(\pi, wx) = \bigcup_{q \in \delta(\pi, w)} \delta(q, x) (\pi \in P(Q), x \in \Sigma, w \in \Sigma^*)$

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