CS 361 Meeting 5 — 2/19/20

Announcements

1. Homework 2 now available. Due Friday.

Closure Properties

1. One 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.

2. Last class, we showed that the complement of any regular language must be regular. That is, we showed that regular languages are closed under set complement.

3. Today, we can give a similar argument to show that the intersection of any two regular languages over the same alphabet must be regular.

   - Suppose that we know that two languages L_1 and L_1 over the same alphabet Σ 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).
   - The key to the correctness of this construction is the fact given the definitions of M_1, M_2 and M, we can show that for all $w \in \Sigma^*$, $\hat{\delta}((q_1, q_2), w) = (\hat{\delta}_1(q_1, w), \hat{\delta}_2(q_2, w))$ using proof by induction on the length of w.

     Base Case: if $|w| = 0$, then $w = \epsilon$. By the base case of the definition of $\hat{\delta}$ we know that $\hat{\delta}(q, \epsilon) = q$ for any δ function. Given the specific definition of δ used in the construction of M in our argument, we can see that $\hat{\delta}((q_1, q_2), \epsilon) = (q_1, q_2) = (\hat{\delta}_1(q_1, \epsilon), \hat{\delta}_2(q_2, \epsilon))$ as required.

     Inductive Case: Assume that

     $$\hat{\delta}((q_1, q_2), w) = (\hat{\delta}_1(q_1, w), \hat{\delta}_2(q_2, w))$$

     for any w such that $|w| \leq n$. Now consider some w' of length $n+1$. We know that we can find a w of length n and a single element $x \in \Sigma$ such that $w' = wx$. Then, we can see that

     $$\begin{array}{lll} \hat{\delta}((q_1,\, q_2), w') = & & \\ & \hat{\delta}((q_1,\, q_2), wx) = & \\ & \delta(\hat{\delta}((q_1, q_2), w), x) = & \text{by def. of } \hat{\delta} \\ & \delta((\hat{\delta}_1(q_1, w), \hat{\delta}_2(q_2, w)), x) = & \text{by ind. assumption} \\ & (\delta_1(\hat{\delta}_1(q_1, w), x), \delta_2(\hat{\delta}_2(q_2, w), x)) = & \text{by def. of } \delta \\ & (\hat{\delta}_1(q_1, wx), \hat{\delta}_2(q_2, wx)) = & \text{by defs. of } \hat{\delta}_1 \text{ \& } \hat{\delta}_2 \\ & (\hat{\delta}_1(q_1, w'), \hat{\delta}_2(q_2, w')) & \end{array}$$
   - From this we can see argue that $w \in L_1 \cap Ł_2 \iff w \in L_1$ 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.

4. 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.

   Proof: Suppose that we know that two languages L_1 and L_1 over the same alphabet Σ 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 $L_2 = L(M_2)$

Click here to view the slides for this class

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 $\delta((q_1, q_2), x) = (\delta_1(q_1, x), \delta_2(q_2, x))$.

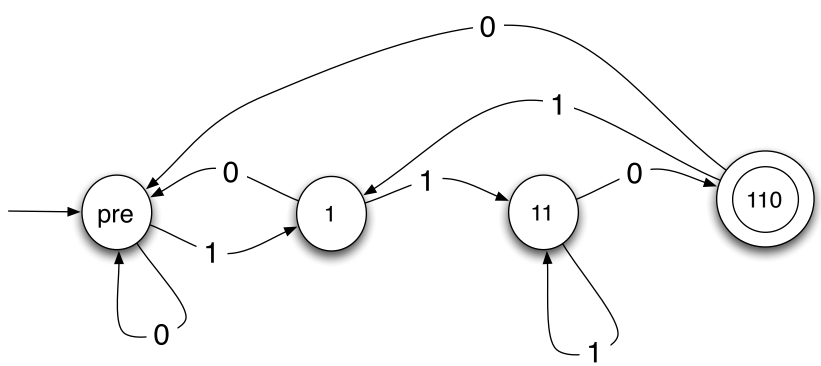
5. Better yet, we can take a simpler approach by observing that $A \cup B = \overline{\bar{A} \cap \bar{B}}$

A Little More Practice

1. Take a moment to work on sketching a DFAs for the following language:

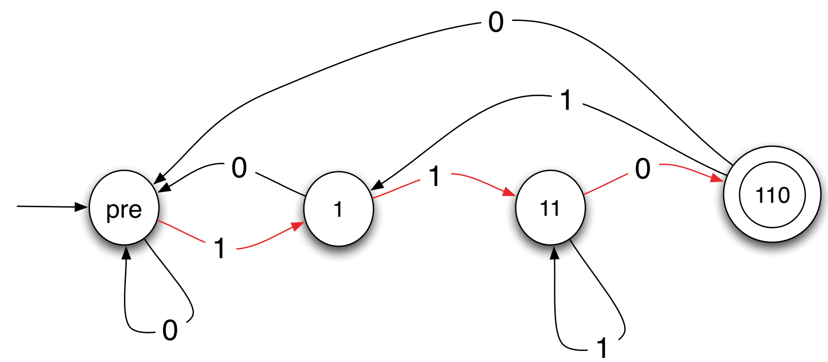
   (a) The language of binary strings ending in 110.

2. I am guessing that the solutions you found to the exercise above look something like:



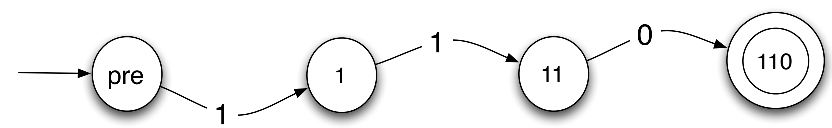
3. Let's consider an alternate approach to describing the language.

   - 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:

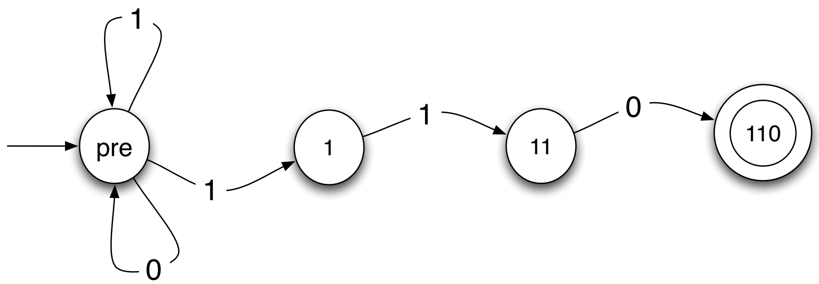


   - Most of the non-red edges in this machine show what to do when something that looked like the final 110 turned out not to be!

   - 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 "ϵ-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
   Σ is the input alphabet
   $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ 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 δ so that we can apply it to strings instead of just individual symbols:

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

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