CS 361 Meeting 6 — 2/21/20

Announcements

1. Homework 2 due today.

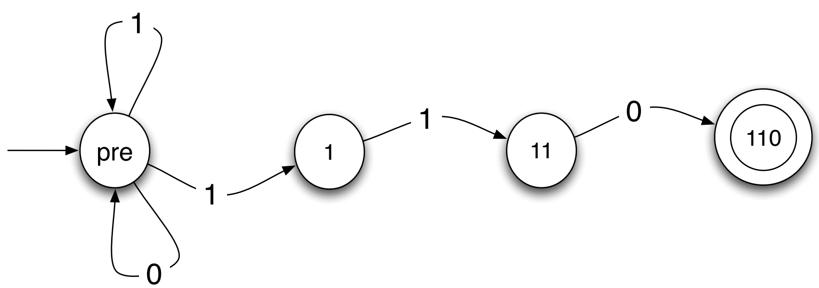
2. Sample solutions for homework 1 are online. These are for current 361 students only. You are expected not to share them with other students who might take the course in the future or to post them anywhere others might find them.

3. Forgive my typos (or even report them) in both these lecture notes, homework solutions and even in my comments on your homeworks.

4. Homework 3 will be posted over the weekend.

Review/Practice

1. Nondeterministic Finite Automata allow multiple possible transitions out of a single state for a single input symbol:

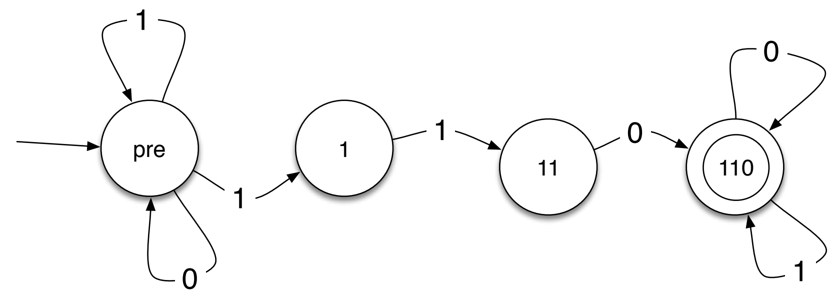


   The machine uses this extension to recognize binary strings that end with 110.

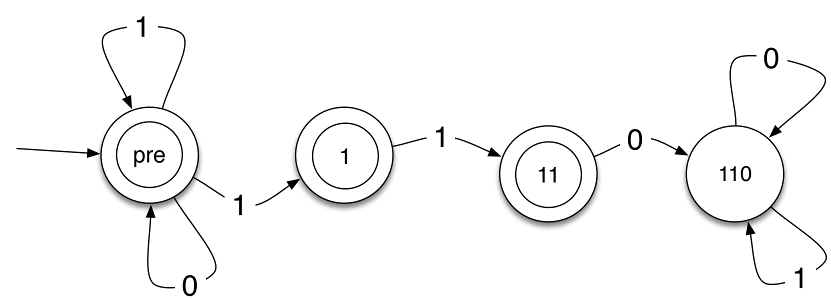
2. Consider two language closely related to the machine shown above (which we discussed last class):

   - $E = \{w | w \in \{0,1\}^*$ and w contains $110\}$
   - $\overline{E} = \{w | w \in \{0,1\}^*$ and w does not contain $110\}$

3. A non-deterministic machine (very similar to the "ends in 110" machine above) for the language E is shown below.



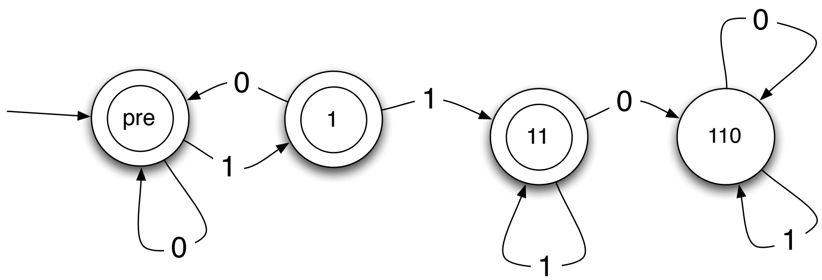
4. The language $\overline{E}$ is just the complement of E. For deterministic finite automata, we argued that you could obtain a machine that recognized the complement of another machine's language by reversing the final and non-final state. Applying this approach to the machine above yields:



5. What language does this machine recognize? It is not $\overline{E}$. Instead, it recognizes $\{0,1\}^*$ because it can just stay in the initial (and final) state on any sequence of 0's and 1's.

6. To recognize $\overline{E}$, we have to resort to determinism:

Click here to view the slides for this class



Formalizing Non-determinism (Review)

1. Just as we gave formal definitions to explain how to understand DFAs, we can do the same for NFAs. **Definition.** A 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

2. The only difference between this definition and the definition of a DFA is the δ returns some (possibly empty) set of states that could be the next state of the machine rather than returning the single state that is the next state.

3. Using this formalism, we can describe the "ends with 110" machine drawn above as

$$M = (\{pre, 1, 11, 110\}, \{0, 1\}, \delta, 0, \{3\})$$

   where

   - $\delta(pre, 0) = \{pre\}$
   - $\delta(pre, 1) = \{pre, 1\}$
   - $\delta(1, 1) = \{11\}$
   - $\delta(11, 0) = \{110\}$
   - and otherwise $\delta(s, d) = \{\}$

Nondeterminism = Determinism?

1. One of the most interesting properties of nondeterministic finite automata is that they are no more powerful than finite automata.

   - Whether you take the parallel or the "great guesser" interpretation of nondeterminism, this should be a little surprising.

2. Our most important goal today will be to explore a proof of this fact.

   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. The basic idea of the proof is that given a NFA N, we can construct a DFA D each of whose states corresponds to some subset of states that might be simultaneously active under the parallel or "follow all possible paths" interpretation of nondeterminism.

4. 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.

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

5. Given this set of states, s_D should be $\{s_N\}$ representing the fact that the nondeterministic machine is limited to the single state s_N when it starts scanning its input.

6. F_D should contain all subsets of states that include any final state of N. That is

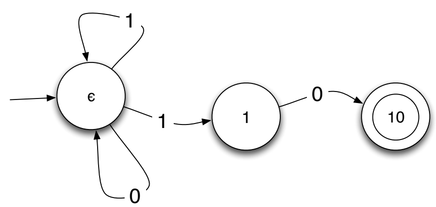
$$F_D = \{\pi \mid \pi \in \mathcal{P}(Q) \text{ and } \pi \cap F_N \neq \emptyset\}$$

   This reflects the idea that if any of the possible paths of computations allowed ends in a final state then the input should be accepted.

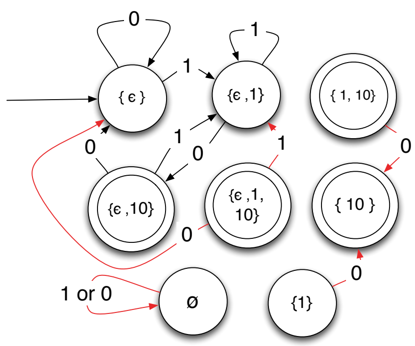
7. δ_D should be defined to figure out where we might go from each state of N in the current state of D. That is:

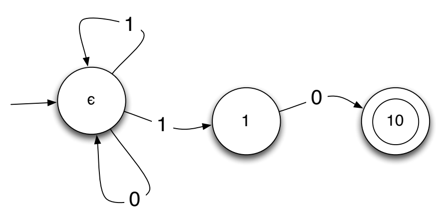
$$\delta_D(\pi, x) = \bigcup_{q \in \pi} \delta_N(q, x)$$

8. To make all this formalism more concrete (and hopefully understandable) let's see what the DFA to simulate the following NFA would look like.



The DFA would look like:





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

- While the set of states includes all subsets of Q, only those subset that can actually be reached from the start state on some input are actually important.
- The edges shown in black in the diagram are the edges that connect reachable states. These are the only edges that will ever actually be used.
- The edges in red show most of the edges between unreachable states that follow from the formal definition. To keep the diagram simple some of these edges (mainly transitions on 1 to the "fail" state associated with the empty set of states) have been omitted.
- Typically, when performing such a construction we would only actually show the reachable states and the connecting edges.

Closure under Reversal

1. Given that DFAs and NFAs are of equivalent power it is often easier to prove languages are regular by designing an appropriate NFA than by defining a DFA. We will now consider another example that both illustrates this and provides motivation for the final feature of the NFA model, ϵ-transitions.

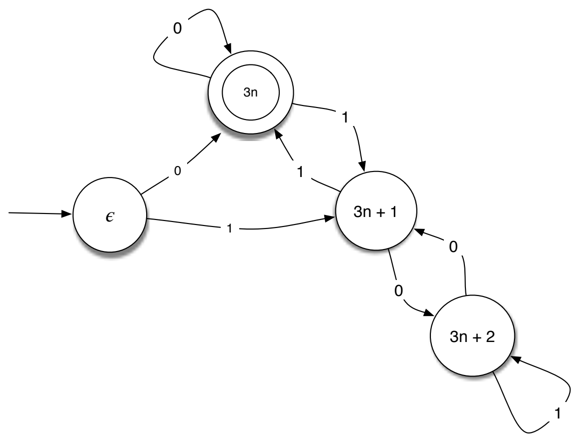
2. Given a language L, we can define L^R to be the language of all strings obtained by reversing the strings in L. If we let w^R represent the reversal of a string w then $L^R = \{w^R \mid w \in L\}$.

3. For homework, I asked you to show that the reversal of the language of binary numbers that were divisible by 3 was also a regular language. This is, in fact, a general property of regular languages. The reversal of any language that is regular is also a regular language.

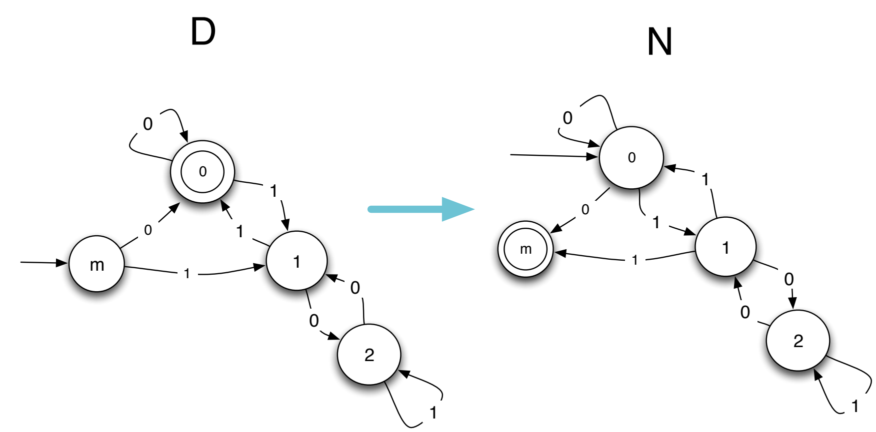
4. The idea behind the proof of this fact is that given a DFA (or even an NFA) M for a language L, we can construct a machine M^R that essentially simulates M running backward. That is, M^R starts in the final state of M (assuming for a moment than M only has one final state) and on any input it follows one of the edges labeled with that input that points to the current state backward to move to a state from which it could have reached the previous state on the input symbol. It accepts if it can find a backward path that leads to the start state.

5. To understand the process a bit better, consider how we would transform the multiples of three machine.

   - The diagram for our multiples-of-3 machine is shown below:

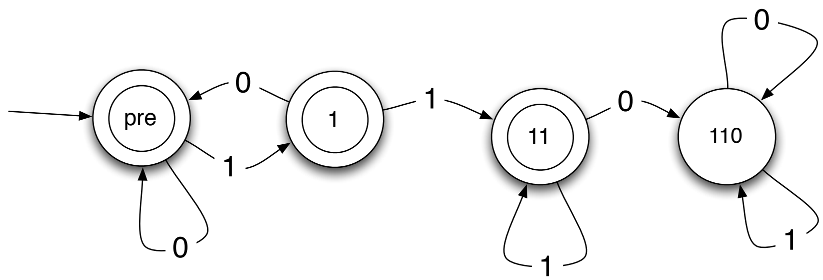


   - Consider the binary number 10010 and how we could use the machine above to realize that this number is divisible by 3 even if we insisted on considering the digits from right to left.
     - We know the number will only be accepted by our machine if after reading all the digits it ends up in state 3n. So, assume this is where the machine will end up.
     - Now, ask what state the machine must have been in before the last digit (which is a 0) to end up in state 3n.
     - Looking at the diagram, there are two ways to get to state 3n after seeing a 0. We must have either already have been in state 3n or have been in the start state, ϵ.
     - Like a good non-deterministic machine, we have to guess. Since we know there are multiple digits to the left of the final 0, we can guess we didn't come from the start state and conclude we must have come from state 3n.
     - No, we know that after reading the next to last digit (a 1), our machine would have to end up in state 3n (if it is going to ultimately accept the input).
     - The only way to get to 3n on a 1 is to come from 3n + 1 so we can assume we were in that state before reading the last two digits.
     - To be in 3n + 1 after reading the third digit from the end (a 0), we must have come from 3n + 2.
     - By similar reasoning, to get to 3n + 2 after the fourth digit from the end (another 0), we must have been in 3n +1 after the first digit.
     - There are two ways we could reach 3n + 1 after reading the first digit, coming from 3n or from ϵ. We have to guess again. Knowing that this is the final digit the right guess is obviously ϵ since this completes the process of finding a path from the start state to a final state on the given input.
   - We can turn this process into an actual finite automaton by simply reversing all of the transition edges in the original machine, making its final state the start state and making its start state the only final state. This transition is illustrated below:

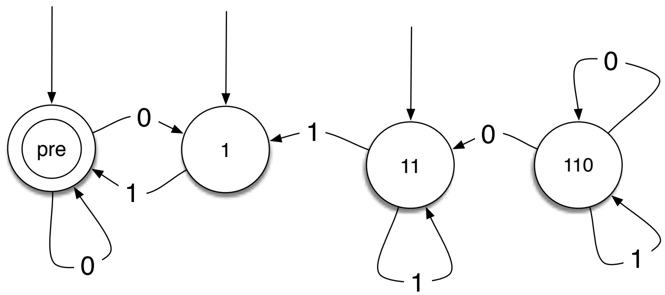


6. To generalize this result to cover finite automata with multiple final states, it helps to incorporate a feature of nondeterministic finite automata that we omitted initially to simplify our presentation: ϵ-transitions.

   - Consider the machine shown above (and repeated below) to recognize binary strings that don't contain the substring 110:



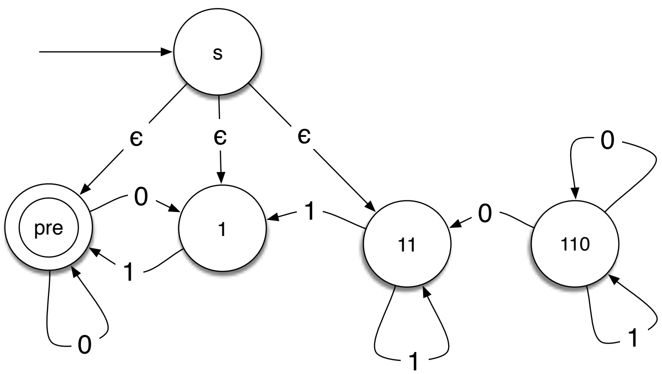
- If we want to derive a machine that recognizes the reverse of this machine's language(binary strings that don't contain the substring 011), we can reverse the transition arrows and make the start state final as we did for the divisible-by-three machine.
- The problem is that the original machine has three final states and we cannot make all of the start states because our definition of finite automata (whether deterministic or nondeterministic) only allows one start state:



- What we really want is to let the machine chose to start in any of these three states. The standard definition of nondeterminisitic finite automata contains a feature that makes this possible.

ϵ-transitions

1. As the name suggestions, ϵ-transitions are transitions that a nondeterministic finite automaton may take while "reading ϵ " from the input stream or more accurately without consuming any symbols from the input.
2. If an NFA reaches state s at some point in its computation and there is an epsilon transition from s to s', then the NFA can move to s' without consuming any input.
3. As a simple example, we can solve the problem with our machine for strings not containing 011 by adding a new start state from which it is allowed to move to any of the former final states of "not containing 110" machine we are trying to reverse:



4. Now that we have seen that ϵ-transitions can be handy when construction a DFA, we need to show how they can be incorporated in the formalism for NFAs.

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
$\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ is a state transition function
$s \in Q$ is the start state
$F \subseteq Q$ is a set of accept states

where $\Sigma_\epsilon = \Sigma \cup \epsilon$.

5. The introduction of Σ_ϵ is the only change in this definition.

6. We also need to adjust our definition of $\hat{\delta}$ to accommodate the change to the transition function. To do this, we first introduce what is called the ϵ-closure of a set of states:

Definition. Let $E : \mathcal{P}(Q) \to \mathcal{P}(Q)$ be defined recursively as:

$$E(\pi) = \{q \mid q \in \pi \text{ or for some } q' \in E(\pi), q = \delta(q', \epsilon)\}$$

7. Our fancy recursive definition is just a way of saying that $E(\pi)$ is the set of all states reachable from any state in π using only ϵ-transitions.

8. Given this definition of the ϵ-closure, we can revise our extension of δ to sets of states and strings as:

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

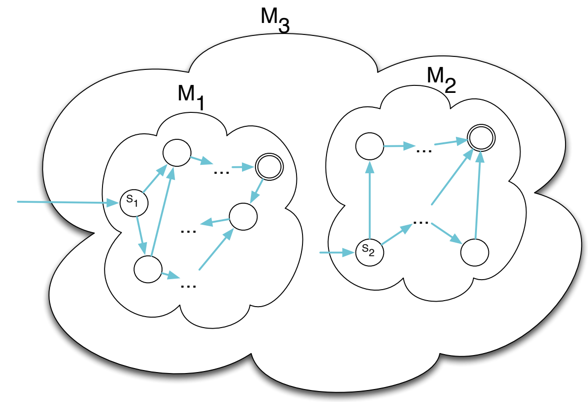
9. Even with ϵ-transitions, NFAs are still equivalent in power to DFAs. That is, for any language L, there is an NFA that recognized L if and only if L is regular.

More closure properties

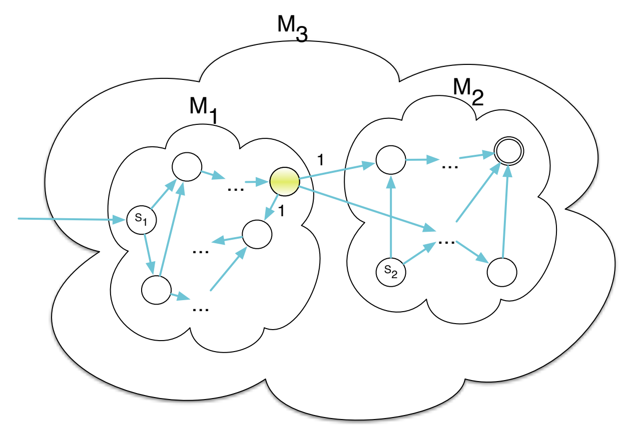
1. When we introduced the notion of a language as a set of strings, we also introduced a product operation specific to sets of strings in which the strings in the product were formed by concatenating two strings from the languages to which the product is applied.

2. 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. M_3 would somehow have copies of M_1 and M_2 inside:

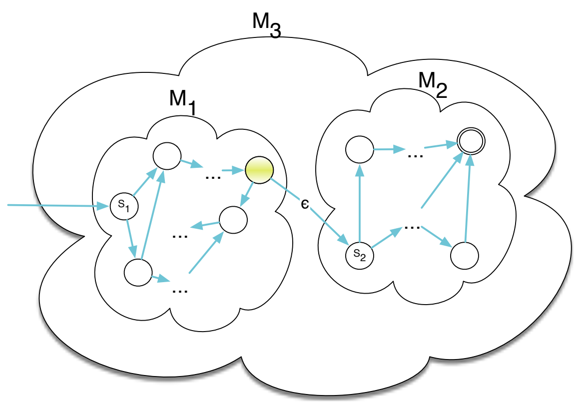


   - If we start in the start state of M_1 and somehow once we get to a final state allow ourselves to act as if we are actually at the start state of M_2, each sub-machine can check that its part of the input string belongs to the right language.
   - To accomplish this, we might add transitions from the final state of M_1 (we are assuming there is just one to keep this intuitive argument simple) to the states reachable from the start state of M_2 to the definition of 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 edges we add will leave the final state of M_1 with two choices for its next state on each input symbols. This requires non-determinism.

3. Given that we are using nondeterminism anyway, we can make things simpler by just having an ϵ transition from each final state of M_1 to the start state of M_2.



4. It is worth noting that this can be formalized by saying that if $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ are DFAs that accept L_1 and L_2 then $L_1L_2 = L(M_3)$ where the NFA M_3 is defined as:

$$M_3 = (Q_1 \cup Q_2, \Sigma, \delta_3, s_1, F_2)$$

with

- $\delta_3(q, \epsilon) = \{s_2\}$ if $q \in F_1$
- $\delta_3(q, x) = \{\delta_2(q, x)\}$ if $q \in Q_2$
- $\delta_3(q, x) = \{\delta_1(q, x)\}$ if $q \in Q_1$
- $\delta_3(q, x) = \{\}$ otherwise