CS 361 Meeting 4 — 2/17/20

Announcements

1. Homework 2 is now online. The material related to the "non-deterministic" components of the last two problems will be covered on Wednesday.

Quick Review

1. Last time, I ended by presenting a mathematical formalism for deterministic finite automata. This will enable us to reason more broadly about their properties.

   Definition. A DFA is a five tuple $M = (Q, \Sigma, \delta, s, F)$ where:

   Q is a finite set of states

   Σ is the input alphabet

   $\delta : Q \times \Sigma \to Q$ is a state transition function

   $s \in Q$ is the start state

   $F \subseteq Q$ is a set of accept states

2. Using this notation, we can give a formal description of our machine that recognizes even binary numbers:

   - $Q = \{e, o\}$
   - $\Sigma = \{0, 1\}$
   -

δ:	0	1
e	e	o
o	e	o

   - $s = o$
   - $F = \{e\}$

Families of Languages and Machines

1. One advantage of this formal definition of a DFA is that it allows us to describe and reason about collections of similar DFAs rather than needing to draw a diagram for a specific DFA.

2. Through our examples, we have seen that the set of strings that represent binary numbers that are even is regular and that the language of binary numbers that are divisible by 3 is regular. We might well speculate that the language

$$L_{DivN} = \{w \mid w \in \{0,1\}^* \text{ and } w \text{ encodes a number divisible by } N\}$$

   is regular for all values of N.

3. To show that all these languages are regular, we will describe a family of DFAs, M_{DivN} such that $L(M_{DivN}) = L_{DivN}$. In particular, we define

$$M_{DivN} = (Q_N, \{0,1\}, \delta_N, m, 0)$$

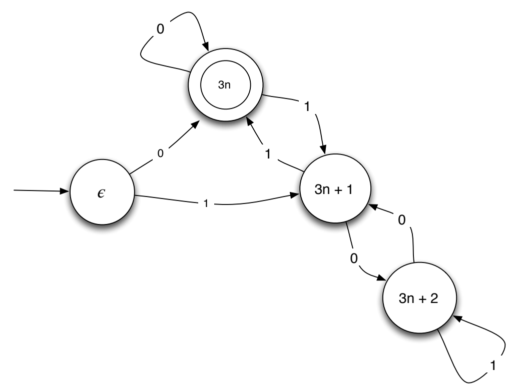
   where:

   Q_N = { m, 0, 1, ... , N-1 } and

   δ_N is defined by

   - $\delta_N(m, 0) = 0$
   - $\delta_N(m, 1) = 1$
   - $\delta_N(i, d) = (2i + d) \bmod N$ for $i \in \{0, 1, ..., N-1\}$

4. You might recognize that the machine presented as a solution to one of our class exercises is M_{Div3}:



Click here to view the slides for this class

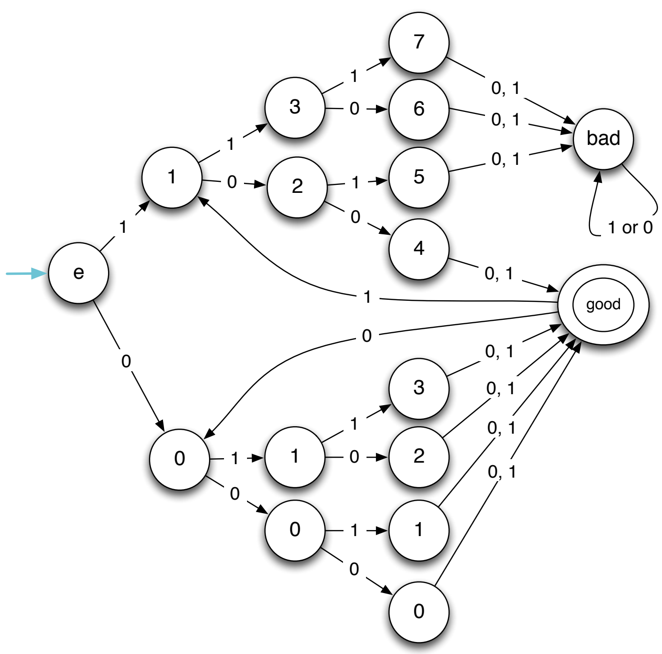
Building State Sets

1. When giving a formal description of a machine, it is useful to think about various ways to describe the state set. Constructing a state set out of tuples of values including subsets of the integers and other setsand can also make it possible to described the transition function δ clearly and concisely using the well known operations like addition.

   - Hopefully, it was evident that we could not have included the line

     $$\delta_N(i, d) = (2i + d) \bmod N \text{ for } i \in \{0, 1, ..., N - 1\}$$

     in our definition of M_{DivN} if we had defined the state set for this machine as $Q = \{$ none, divisible, remainderis1, remainderis2$\}$.
   - Another example that illustrates how useful it can be to include subsets of the integers in your state set is the BCD example we considered.
   - I presented an FSA that solves this problem by tracking the value of the prefix of the group of four bits currently being scanned:



   - We could take this idea even farther and replace "good" by 10 states labeled with the values 0 through 9 and "bad" with 6 states labeled with 10 through 15.
   - The transitions out of each of the states in this diagram depend on the value written in the center of the state and the column in the diagram in which the state occurs. The column each state falls in corresponds to the position the machine is up to in the group of four digits currently being scanned.
   - Based on this observation, we can build a state space out of pairs of integers. The first integer in each pair will encode the column each state in the diagram appears in, the second integer will encode the value of the prefix of the group of four digits that has been scanned so far.
   - Given this approach, we can define $M_{BCD} = (Q, \Sigma, \delta, s, F)$ where:
     - $\Sigma = \{0, 1\}$
     - $Q = \{0, 1, 2, 3, 4\} \times \{0, ...15\}$
     - $s = (0, 0)$
     - $F = \{4\} \times \{0, \ldots, 9\}$
     - $\delta((c, v), i) = (c + 1, 2v + i)$, if $c < 4$
     - $\delta((c, v), i) = (1, i)$, if $c = 4$ & $v < 10$
     - $\delta((c, v), i) = (c, v)$, if $c = 4$ & $v \geq 10$

Formalizing L(M)

1. Earlier we gave simple rules for evaluating a string with respect to the diagram of a finite automaton. Now we can give a formal definition of how to decide if an automaton accepts or rejects a string.

   - Our text defines the notion that a string belongs to the language of a DFA in terms of the existence of a sequence of states related to the sequence of symbols in the string and the machine's definition in ways that reflect the intent to capture transitions with the δ function.

     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 $1 \leq i \leq n$
  - $q_n \in F$

- We will frequently use an alternate but equivalent definition that depends on a recursive definition of a function that extends δ.
- Let $\hat{\delta}$ be a function from $Q \times \Sigma^*$ to Q (instead of just $Q \times \Sigma$ like δ). That is, it operates on state/string pairs instead of just state/character pairs.
- We want $\hat{\delta}$ to function as an extension of δ 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 δ:

$$\begin{array}{rcll} \hat{\delta}(q, \varepsilon) & = & q & (q \in Q) \\ \hat{\delta}(q, wx) & = & \delta(\hat{\delta}(q, w), x) & (q \in Q,\, x \in \Sigma,\, w \in \Sigma^*) \end{array}$$

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

2. With $\hat{\delta}$ in hand we can give a definition of acceptance:

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

3. The language of a DFA M follows naturally:

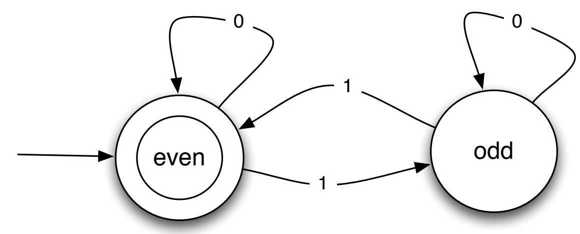
   Definition $L(M) = \{w \mid w \in \Sigma^* \text{ and } \hat{\delta}(s, w) \in F\}$

Closure Properties

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

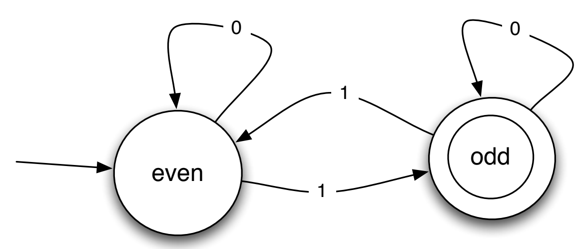
   - We say that a set is closed under an operation if performing the operation on elements of the set only produces other elements of the set. The integers, for example, are closed under addition but not under division.
   - Since languages are just sets, it is interesting to ask whether the set of regular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

2. Consider the DFA we showed above that recognized 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 Σ^* then the language of all strings in Σ^* not in L is also regular.

Our formalism gives us a means to prove this:

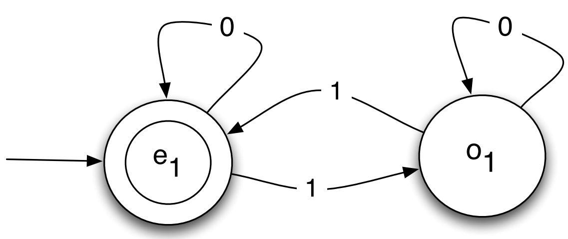
Theorem: For any language L over an alphabet Σ^*, the complement of L, $\bar{L}$ is also a regular language.

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

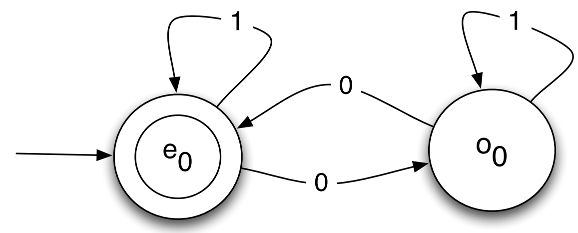
Let $M' = (Q, \Sigma, \delta, q_0, Q - F)$. Now we know that $w \in \bar{L} \iff w \notin L \iff w \notin L(M) \iff \hat{\delta}(q_0, w) \notin F \iff \hat{\delta}(q_0, w) \in Q - F \iff w \in L(M')$. This shows that $\bar{L} = L(M')$ so we can conclude that $\bar{L}$ is regular.

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 languages) we can conclude that the language of strings containing even numbers of both 0s and 1s is regular.