CSE547T Class 3

Jeremy Buhler

January 25, 2017

1 DFAs are a pain!

You may have noticed that DFAs are kind of tricky to work with.

- Must have transitions from all states on all chars
- Not easy to deal with "recurrent" patterns
- Is there a machine that isn't quite so yucky?
- What we're lacking is *expressiveness*.
- (think C++ vs assembly)

Here's a trick that should greatly increase our expressiveness.

- A key aspect of our DFAs is that they are **deterministic**.
- Given state and next character, the DFA moves to *exactly one* next state.
- We will relax this restriction to allow the machine to make multiple moves from each state.
- (Note: this is not terribly intuitive for real computers, but it is related to backtracking search)

2 Definition of NFAs

The new type of machine is called a *nondeterministic finite automaton* (NFA).

- **Defn**: an NFA is a 5-tuple $M = (Q, \Sigma, q_0, A, \delta)$, where
- Q is a finite set of states
- Σ is the alphabet of input chars
- $q_0 \in Q$ is the unique starting state
- $A \subseteq Q$ is the set of accepting states
- $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function.

- If M in state q and sees char a, it *nondeterministically* chooses a next state from the set $\delta(q, a)$.
- (Another way to think of δ is that the NFA duplicates itself $|\delta(q, a)|$ times, and each continues starting with one of the possible moves until the input string runs out.)

OK, example time. Here is our automaton for recognizing strings containing aba, this time as an NFA.

Some interesting features of this new machine...

- Ugly backwards edges are all gone. Where did they go?
- Machine nondeterministically guesses when to start matching aba.
- If it guesses right, it will get to the accepting state.
- Otherwise, it will *crash and burn*.

Important: we need to define δ^* and acceptance for NFAs. They aren't quite the same as for DFAs.

- **Defn**: For an NFA M, the extended transition function δ^* is defined as follows.
- For any $q \in Q$, $\delta^*(q, \varepsilon) = \{q\}$. (Notice the curly "set braces" around q).
- For any $q \in Q$ and string $x = ya$,

$$\delta^*(q, x) = \bigcup_{r \in \delta^*(q,y)} \delta(r, a).$$

- Alternatively, if $x = bz$,

$$\delta^*(q, x) = \bigcup_{r \in \delta(q,b)} \delta^*(r, z).$$

I hate big unions, so here is some simplifying notation.

- For any set of states ψ of an NFA and any $a \in \Sigma$, define

$$\delta(\psi, a) = \bigcup_{q \in \psi} \delta(q, a).$$

- Now we can write for any $x = ya$,

$$\delta^*(q, x) = \delta(\delta^*(q, y), a).$$

- We can define similar notation to simplify δ^* for NFAs:

$$\delta^*(\psi, x) = \bigcup_{q \in \psi} \delta^*(q, x).$$

- Under this new notation, can prove from old defns (**Exercise!**) that

$$\delta^*(\psi, \varepsilon) = \psi$$

  and, for $x = ya$, that

$$\delta^*(\psi, x) = \delta(\delta^*(\psi, y), a).$$

- Or, turning it around, we can show that for $x = bz$,

$$\delta^*(\psi, x) = \delta^*(\delta(\psi, b), z).$$

- Hence, our fancy new δ^* on sets behaves just like the boring old δ^* on individual states.

Let's see what the above means for our string-accepting machine.

- Try $\delta^*(q_\varepsilon, aab)$.

- $$\begin{aligned} \delta^*(q_\varepsilon, aab) &= \delta^*(\{q_\varepsilon, q_a\}, ab) \\ &= \delta^*(\{q_\varepsilon, q_a\} \cup \{q_{ab}\}, b) \\ &= (\{q_\varepsilon\} \cup \{q_{ab}\}) \cup \emptyset \\ &= \{q_\varepsilon, q_{ab}\}. \end{aligned}$$

- Now what if the next input char is an a?

- We end up in $\{q_\varepsilon, q_a, q_{aba}\}$. Did we accept?

- We have proven that there *exists* a set of choices that would get M from q_0 to an accepting state on $x = aaba$.

- **Defn**: an NFA M accepts a string $x \in \Sigma^*$ if

$$\delta^*(q_0, x) \cap A \neq \emptyset.$$

  That is, M accepts if *at least one* series of nondeterministic choices will get it from q_0 to A on input x.

- So yes, the NFA does accept $aaba$.

One more note: just as before, we can use tabular notation to describe δ for NFAs. It looks about like you'd expect.

3 NFAs are Cool

Let's see another example of the expressive power of NFAs.

- Let $L = \{x \in \{0,1\}^* \mid x \text{ is of the form } \{11, 110\}^*0\}$.
- Here's an NFA M that I claim accepts L:

- How sure are you that this NFA is right? Let's prove it (quickly)
- ($\rightarrow$) suppose $x \in L$. Then x can be subdivided into zero or more copies of the strings 11 and 110, followed by a 0.
- Given this subdivision, I claim that there is an accepting path for x in M. In particular, starting from q_0, follow the top loop for each instance of 110 and the bottom loop for each instance of 11, then use the final 0 to reach the accepting state.
- ($\leftarrow$) Now suppose $x \in L(M)$. Any accepting computation follows a path from the start state to the single accepting state.
- All such paths must end with the edge from q_0 to q_a, so the strings that trace them have the form $y0$ for some y.
- Note that the path in M corresponding to string y must start and end at q_0.

- I claim that all paths in M starting and ending at q_0 correspond to strings $y \in \{11, 110\}^*$.
- Prove by induction on path length.
- (**Bas**): The shortest path from q_0 to q_0 is the null path. In this case, $y = \varepsilon$, which has the desired form.
- (**Ind**): suppose claim holds for all path lengths $< n$. Consider a path of length n.
- Since path must end at q_0, it must have come in via either q_{12} or q_2.
- By the graph structure and the fact that the path starts at q_0, the last few steps on the path must be either q_0, q_{11}, q_{12}, q_0 or q_0, q_2, q_0.
- These cycles respectively imply that the string y ends with 110 or 11.
- The remaining prefix of the path also starts and ends at q_0. By IH, it must correspond to a string $z \in \{11, 110\}^*$.
- Conclude that either $y = z11$ or $y = z110$, and hence y has the desired form. Hence, $x \in L$. QED

Want to try recognizing this with a DFA? I think it's pretty hard. Exercise 1: build it. Exercise 2: prove it's right!