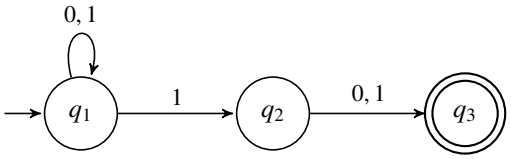
Lecture 5: NFA vs. DFA

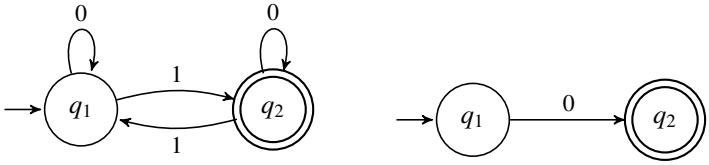
built on 2021/01/16 at 16:19:51

In the previous lecture, we learned about nondeterminism—in particular, finite-state machines with nondeterminism. Structurally, every DFA is by definition an NFA. How about power? Are NFA—because of nondeterminism—strictly more powerful than DFA?

There is ample evidence suggesting that the NFA may be more powerful—at least more expressive—than their DFA counterparts. For example, if we're to design a DFA that recognizes all binary strings whose 2nd position from the end is a 1, we know from your Assignment 1 that the DFA will have at least 4 states. However, the following NFA does the job using only *three* states.



Moreover, the analog of the union theorem on NFAs appears much simpler. If N_1 and N_2 are NFAs recognizing the languages L_1 and L_2, what's an NFA that recognizes $L_1 \cup L_2$ going to look like? Let's first look at a few examples. On your left below is a machine that recognizes all binary strings with an odd number of 1s. On your right is a machine that accepts if the input is a 0.



The union of the two languages is simply either the input is 0 or the input has an odd number of 1s. What does such an NFA look like?

The answer is simpler than you might think initially. Simply draw them together as one machine—no changes otherwise.

The surprising thing is that as it turns out, the NFA and DFA are equally powerful: every NFA has a DFA that simulates it.

1 Equivalence Between DFA and NFA

Specifically, we prove the following theorem. We say that two machines N and M are equivalent if $L(M) = L(N)$.

Theorem 1.1 *Every NFA has an equivalent DFA.*

Before we go on to sketch a proof, let us look at two direct consequences:

(1) a language is regular $\iff$ it is recognized by an NFA;
(2) a language L is regular if and only if L^{R} is regular.

1.1 From NFA to DFA

To prove the theorem, we'll take as input an NFA $N = (Q, \Sigma, \delta, Q_0, F)$ and construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ that recognizes the same language as the NFA N. That is, we're converting an NFA into an equivalent DFA.

How could we do such a thing? One crucial difference between NFA and DFA is that in an NFA, you can be in multiple states at once. We could simulate this situation by making each DFA state represent a set of NFA states.

For instance, if you are in states q_1, q_2, q_7 in an NFA, you would be in the state $\{q_1, q_2, q_7\}$—yup, a state labeled with a set. Therefore, we'll use $Q' = 2^Q$. Once we take this view, transition or other things are simple.

Proof:

- Let $Q' = 2^Q$. That is, every state of M is a set of states of N. Because an NFA has finitely many states, Q' still has a finite number of states.
- Let $F' = \{R \in Q' \mid R \cap F \neq \varnothing\}$—in words, a state (remember it's a set) is accepting if it contains an accepting state of the original NFA.

In the absence of ε, it is immediate that the new starting state is the state representing the set Q_0, that is, $q_0' = Q_0$. Furthermore, the transition function is also pretty straightforward: if the NFA is currently in the set of states R and receives a symbol σ, then the transition will cause each $r \in R$ to go to $\delta(r, \sigma)$. Therefore, in the new machine, we have, for $R \in Q'$,

$$\delta'(R, \sigma) = \bigcup_{r \in R} \delta(r, \sigma).$$

The existence of ε somewhat complicates the conversion. Remember that ε is a shortcut—a wormhole that teleports you to places without any symbol at all. Define the ε-closure as follows: For $R \subseteq Q$,

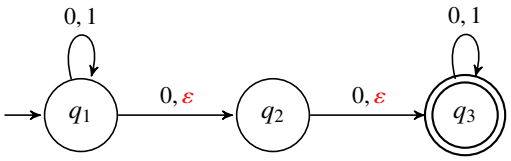
$$\mathcal{E}(R) = \{q \text{ that can be reached from some } r \in R \text{ through 0 or more } \varepsilon \text{ arrows}\}.$$

With this, we can revise our q_0' and δ' as follows:

$$\delta'(R, \sigma) = \bigcup_{r \in R} \mathcal{E}(\delta(r, \sigma))$$
$$q_0' = \mathcal{E}(Q_0)$$

■

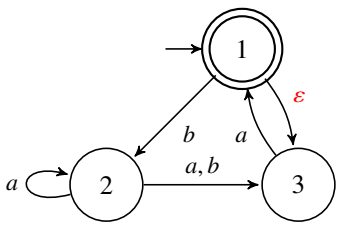
What the heck is ε-closure again? To better understand the idea of the ε-closure. Let's look at an example. Consider the following NFA:



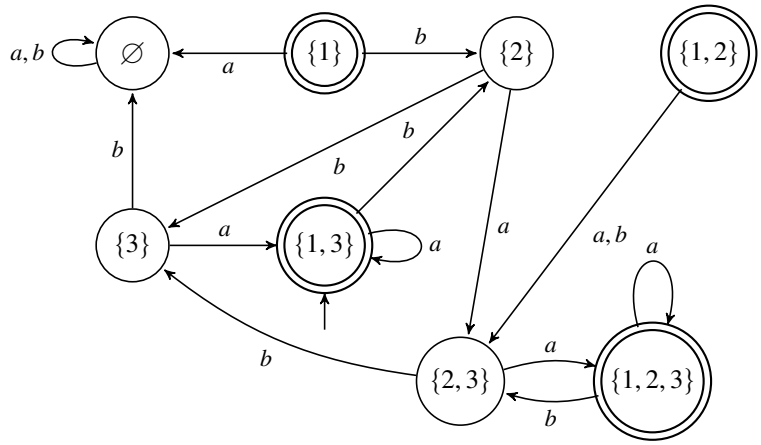
What's $\mathcal{E}(\{q_1\})$? How about $\mathcal{E}(\{q_2\})$? How about $\mathcal{E}(\{q_3\})$? Then, how about $\mathcal{E}(\{q_1, q_2\})$? As it turns out:

$$\mathcal{E}(\{q_1\}) = \{q_1, q_2, q_3\}$$
$$\mathcal{E}(\{q_2\}) = \{q_2, q_3\}$$
$$\mathcal{E}(\{q_3\}) = \{q_3\}$$
$$\mathcal{E}(\{q_1, q_2\}) = \{q_1, q_2, q_3\}$$

Proof Explanation by Example: To gain a better understanding of the proof above, let's see what happens when we apply this construction to the following NFA. Remember that the alphabet is $\Sigma = \{a, b\}$ and the states are $Q = \{1, 2, 3\}$. This therefore leads to an 8-state DFA.



which translates to the following NFA:



Notice that some states can be eliminated without any harm: they are never reached anyway. Specifically, we can get rid of $\{1\}$ and $\{1,2\}$.

Moral: An important reason for introducing NFA is because they can make proofs much easier.

2 Other Regular Operations

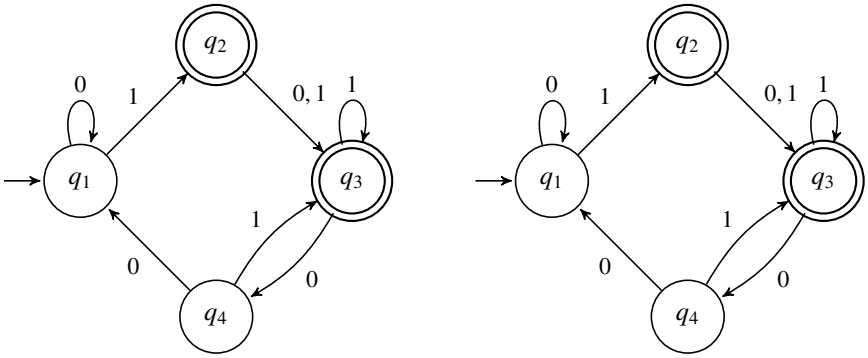
So far we have seen that regular languages are closed under union, interesection, negation, and reverse. There are a few other operations that preserve being regular.

2.1 Concatenation

The concatenation of strings a and b, written $a \cdot b$, is simply the string a followed directly by b. Generalizing this idea, we have that if L_1 and L_2 are languages, then

$$L_1 \cdot L_2 = \{a \cdot b \mid a \in L_1 \wedge b \in L_2\}$$

Let's start with an example. Consider the following DFA:



The proof strategy would still be to build a new machine out of M_1 (for L_1) and M_2 (for L_2). We have a new tool at our disposal: the NFA. To do this, we can simply connect the accepting states of M_1 to the starting state of M_2 using ε; we'll then change the original accepting states of M_1 to be nonaccepting. Formal proof: left as exercise to the reader.

2.2 Star

You have seen the star operation before: Σ^*, for example, is the language of all strings using symbols from Σ. We can generalize this as follows:

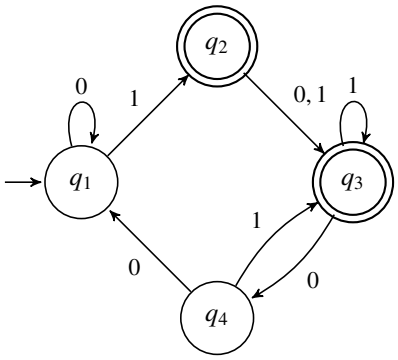
For a language L, a string $a \in L^*$ if a can be formed by putting together any number of strings in L.
More formally, a can be written as $a = w_1 w_2 \dots w_k$, $k \geqslant 0$, where $w_i \in L$ for all $i = 1, \dots, k$.

This means L^* always includes the empty string. As a more concrete example, consider the language $L = \{\texttt{fun}, \texttt{party}\}$. L^* is the language $\{\varepsilon, \texttt{fun}, \texttt{party}, \texttt{funfun}, \texttt{funparty}, \texttt{partyparty}, \texttt{partyfun}, \dots\}$.

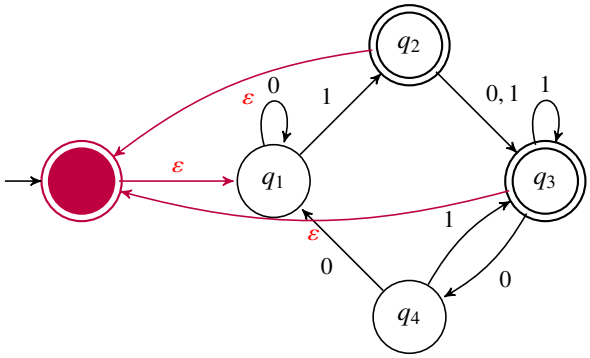
We wish to prove the following theorem:

Theorem 2.1 *If L is a regular language, then L^* is also a regular language.*

Let M be a DFA for L. Say M looks as follows:

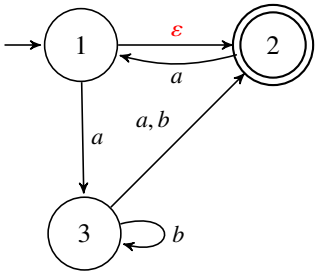


We want two things out of this: ε is accepted, and after you are at an accepting state, you kind of go back to the start. Thanks to NFA and ε, we can readily do this:



3 Practice

Convert the following NFA to an equivalent DFA:



Also, if $L = L(N)$ is the language of the machine N above, construct a DFA/NFA for $L \cdot L$ and another DFA/NFA for L^*.