More About NFA’s, Including $\epsilon$-transitions

Jeffrey Shallit
School of Computer Science
University of Waterloo
shallit@uwaterloo.ca
https://cs.uwaterloo.ca/~shallit
A corollary

We say two automata $M_1$, $M_2$ are *equivalent* if $L(M_1) = L(M_2)$.

The subset construction now proves the following result:

**Theorem.** For every $n$-state NFA $M$, there is an equivalent DFA with at most $2^n$ states.

**Proof.** Apply the subset construction to $M$. 
Let’s carry out the NFA to DFA construction (the “subset construction”) on a simple NFA. Although the formal definition of the subset construction requires $2^n$ states for every NFA of $n$ states, in practice it is easier and usually faster to start with the start state of the NFA, and only construct states that are reachable from this start state. Let’s do it on the following NFA:
The corresponding DFA

The NFA:

The equivalent DFA:
Remember that we extended the DFA model to allow multiple transitions on the same symbol; this gives an NFA.

Now we want to extend it one more time: allowing $\epsilon$-transitions.

If present, an $\epsilon$-transition allows an NFA to spontaneously transit from one state to another, without consuming any symbols of the input.

We denote this, for example, by $\delta(p, \epsilon) = \{q, r\}$, and denote it graphically as follows:
An $\epsilon$-transition does not mean that the automaton is “reading the symbol $\epsilon$ from the input”. There are no $\epsilon$’s in the input! The input is a string of symbols.

It is just a notational convention, to suggest that we can go from one state to the other without reading any input symbols.
Why $\epsilon$-transitions?

This new model is called an $\epsilon$-NFA.

Before we show that this addition to the model doesn’t change the class of languages accepted, let’s see why we would want it.

The virtue of allowing $\epsilon$-transitions is that you can use it to carry out various constructions on automata.

For example, suppose we have NFA’s $M_1, M_2$ for languages $L_1, L_2$. Using these as building blocks, we want to create an NFA for $L_1L_2$.

With $\epsilon$-transitions we can carry this out easily: just put an $\epsilon$ transition from every final state of $M_1$ to the initial state of $M_2$, and have the final states be only the final states of $M_2$. 

Using $\epsilon$-transitions for language concatenation

NFA for $L_1$

NFA for $L_2$

$\epsilon$-NFA for $L_1L_2$
Simulating $\epsilon$-transitions

Let us show (somewhat informally) that NFA with $\epsilon$-transitions can be simulated by ordinary NFA. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an $\epsilon$-NFA. Idea: redefine transitions on a single symbol.

The idea is pretty simple. In an $\epsilon$-NFA, a single symbol $a$ can cause the automaton to enter a large number of states: namely, the states along any path with label

$$\epsilon, \epsilon, \ldots, \epsilon, a, \epsilon, \epsilon, \ldots \epsilon.$$  

So to simulate this with an NFA, we should define the transition function of the NFA, on input $a$, to go to all of the states along the path labeled as above.

This almost works, but not quite. Can you see what goes wrong?
Simulating $\epsilon$-transitions

What goes wrong is that this idea doesn’t quite handle the case when the original $\epsilon$-NFA has a transition $\delta(q_0, \epsilon) = \{q_1\}$, and $q_0$ is nonaccepting and $q_1$ is accepting.

In this case, our simulation as discussed on the previous slide would wrongly reject $\epsilon$.

The solution is to make $q_0$ accepting in the new NFA if it was accepting in the old $\epsilon$-NFA, or if there is an $\epsilon$-labeled path from $q_0$ to an accepting state.
More formally, for $a \in \Sigma$ and $p \in Q$ define a new transition function

$$\delta'(p, a) = \{ q : \text{ there is a path from } p \text{ to } q \text{ labeled } a,$$

$$\quad \text{with an arbitrary number of } \epsilon\text{-transitions before and after} \},$$

and don't include any $\epsilon$-transitions in $\delta'$.

Let

$$F' = \begin{cases} F \cup \{q_0\}, & \text{if } q_0 \not\in F \text{ and there is an} \\
F, & \text{otherwise} \end{cases}$$

Then $M' = (Q, \Sigma, \delta', q_0, F')$ is an ordinary NFA that simulates the $\epsilon$-NFA.
Allowing $\epsilon$-transitions doesn’t change class of languages accepted

A complete formal proof is somewhat long, boring, and clumsy, so we omit it.

So now we know that the classes of languages recognized by DFA’s, NFA’s, and $\epsilon$-NFA’s are all the same: the class $\mathsf{REG}$. A great result!

In proofs, you can always use any of these three, whatever makes the proof easiest.