CSE 322 - Introduction to Formal Methods in Computer Science
Equivalence of DFAs and NFAs

Dave Bacon
Department of Computer Science & Engineering, University of Washington

Last time we defined nondeterministic finite automata. Our motivation for doing this, way back when, was in constructing a DFA which accepted the concatenation of two languages recognized by two different DFAs. Recall that the problem we were considering was to construction a machine which accepted the concatenation of the languages of the two machines M_1 and M_2: Using NFA it is easy to see how to do this: simply take the accept states of M_1

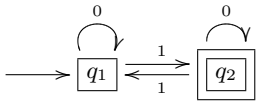

FIG. 1: Machine M_1

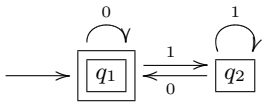

FIG. 2: Machine M_2

and connect them via an ε transition to the states of machine M_2, and make them no longer accept states: Great, so

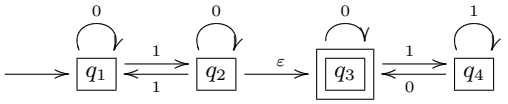

FIG. 3: Machine N

it was simple to construct a NFA which accepts the concatenation of the languages of the two machines. But what good does this do us? NFAs aren't real machines, I mean! They're just crazy, what with their multiple states at one time and states dying out and all. Aha! But we will show, in this section, a method for converting any NFA into a DFA.

Now it is quite clear, even though I forgot to mention it before, that DFAs are subsets of NFAs. If you simply define an NFA with no ε transitions and which the range of every transition function is a set of cardinality 1. Thus after we show that every NFA can be turned into a DFA we will have shown that the languages accepted by DFAs and those accepted by NFAs are exactly the same. In other words NFAs recognize regular languages.

I. THE SUBSET CONSTRUCTION IDEA

Suppose you were trying to run the NFA described by the state diagram in Figure **??**. One way you might do this is by keeping a finger on every state which is currently "alive" at one step during the process. As you progress through evaluating the NFA on a particular input, which states are covered by your fingers will change, of course. But at anytime you will either have the state alive or dead. This will be the key to understanding how we can get a DFA to simulate a NFA. In particular you can already sort of see that if we just keep track of the subsets which we could reach, then we can make a DFA which simulates the action of the NFA we are interested in. There are, however, a few subtleties in this construction, however, not the last of which are the existence of ε labeled transitions. Thus to

begin with let's consider the case where none of the transitions in the NFA we are simulating with a DFA is labeled by ε.

Let $N = (Q, \Sigma, \delta, q_0, F)$ be a NFA recognizing some language $L(N)$ (and further, for now, assume that there are no transitions labeled by ε.) We will now show how to construct a DFA which recognizes this same language $L(N)$. This DFA is described by $M = (Q', \Sigma, \delta', q_0', F')$ and is described via the definitions:

1. $Q' = P(Q)$. Recall that $P(Q)$ is the power set of Q: the set of all subsets of Q. Thus the states of our DFA are made up of subsets of the states of the NFA.

2. For $R \in Q'$ and $a \in \Sigma$ let the transition function be

$$\delta'(R, a) = \{q \in Q | q \in \delta(r, a) \text{ for some } r \in R\} \tag{1}$$

   In other words, the transition function takes the current set of states that is alive and then contains all states that will be transitioned to given the newly read element of the alphabet, a.

3. $q_0' = \{q_0\}$. The start state is the set consisting only of the start state.

4. $F' = \{R \in Q' | R \text{ contains an accept state of } N\}$. In other words the machine accepts if it is a subset of Q that contains an accept state. Note that this includes subsets that include all sorts of reject states, just as long as they have one accept state.

Okay, so informally we can see why the above construction of a DFA from an NFA simulates the action of the NFA: it just carries out the procedure we would have done by keeping our fingers on the states which are currently alive at any one point during the computation. A more formal proof would proceed by induction on the number of steps in the computation.

But wait, we still have to take care of those pesky ε transitions. (Sidenote: Often you will find that NFAs are defined without ε transitions, and then NFAs which use ε transitions are called ε-NFAs. We won't do that and will just allow ε transition in our NFAs.) To take care of the ε transitions it is useful to define $E(R)$ which is the set of states from a machine M which are reachable from the set of states R by traveling only along ε transitions (including R itself.) In other, more formal words, for $R \subseteq Q$ define

$$E(R) = \{q | q \text{ can be reached from the states in } R \text{ by traveling along 0 or more } \epsilon \text{ arrows}\} \tag{2}$$

Thus $E(R)$ captures the notion that if you enter into the states R from some transition of a NFA, then really you are entering into $E(R)$ because of all of the ε transition. We will use $E(R)$ to modify the transition function as follows:

$$\delta'(R, a) = \{q \in Q | q \in E(\delta(r, a)) \text{ for some } r \in R\} \tag{3}$$

Finally we also need to modify the notion of a start state due to the ε transitions. In particular, the start state should now follow all the ε transitions. We can achieve this by letting $q_0' = E(\{q_0\})$. A little bit of contemplating will verify for you that this new definition of the update rule takes care of the ε transitions.

Thus, at least at a fairly informal level, we have shown how it is possible to construct a DFA corresponding to any NFA such that this DFA accepts a string iff the NFA accepts the string. Further, since every DFA is an NFA, what we have shown is that the languages accepted by DFAs is exactly the same as the languages accepted by the NFA. In other words

> A language is regular if and only if some nondeterministic finite automata recognizes it.

That's a pretty cool theorem when you think about it. Even though NFAs seem like a crazy model of computation, it turns out that the languages recognizable by NFAs is exactly that of DFAs.

Of course at this point it is probably useful to think about some of the advantages and disadvantages of DFAs versus NFAs. On advantage of an NFA is that it is often easier to construct an NFA for a language. However, since the NFA is kind of crazy, its not at all obvious how to program a real machine to act like one. However, since we can take an NFA and convert it to a DFA, we can use the NFA we so easily constructed, convert it to a DFA, and then use this DFA if we need to actually code up a machine which accepts the language we are interested in. Note however one bad aspect of this process: if a DFA has $|Q|$ states, then the subset construction given above has $|P(Q)| = 2^{|Q|}$ states. This is an exponential blowup. Later we will talk about minimizing DFAs: but right now this blowup is rather severe and a bit of an obstacle in using this for NFAs which are large.

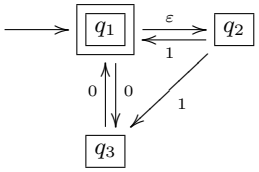

FIG. 4: Machine N_{ex}

II. EXAMPLE OF CONVERTING A NFA TO A DFA

Let's work an example, just for fun (what else!), for converting between a NFA and a DFA. In Figure **??** we show a NFA. So first of all we need to construct the states of the DFA which will be equivalent to this NFA. To do this we construct the power set of the NFAs states: $Q = \{q_1, q_2, q_3\}$, so $P(Q) = \{\emptyset, \{q_1\}, \{q_2\}, \{q_3\}, \{q_1, q_2\}, \{q_1, q_3\}, \{q_2, q_3\}, \{q_1, q_2, q_3\}\}$. It is useful, then, we figuring out the DFA, to construct the $E(R)$ for each of these subsets

$$\begin{aligned} E(\emptyset) &= \emptyset \\ E(\{q_1\}) &= \{q_1, q_2\} \\ E(\{q_2\}) &= \{q_2\} \\ E(\{q_3\}) &= \{q_3\} \\ E(\{q_1, q_2\}) &= \{q_1, q_2\} \\ E(\{q_1, q_3\}) &= \{q_1, q_2, q_3\} \\ E(\{q_2, q_3\}) &= \{q_2, q_3\} \\ E(\{q_1, q_2, q_3\}) &= \{q_1, q_2, q_3\} \end{aligned} \tag{4}$$

Now we can use the NFA to evaluate how we transition among our new DFA, making sure to use the E function above to evaluate the ε transitions. We have, in constructing this state diagram, used the fact that the start state of the

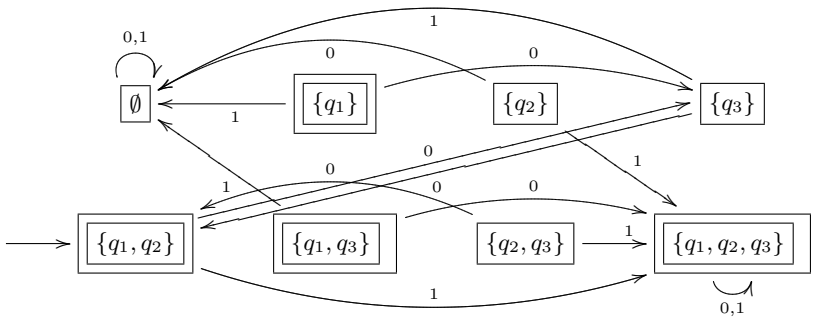

FIG. 5: DFA corresponding to N_{ex}

NFA, q_1 transitions via a ε transition to $\{q_1, q_2\}$, and also the fact that the accept state is any subset that contains q_1.

Note that $\{q_1\}$ and $\{q_1, q_3\}$ is never in the range of E. This implies that the NFA will never enter these states. This is why writing down the E function before hand helps. You can immediately leave out the subsets which will never be reached. Further note that $\{q_2, q_3\}$ and $\{q_2\}$ have no incoming arrows. Thus they can also be eliminated. Thus we could reduce this to From this figure it is easy to see that the language of this machine is the empty string, any string that begins with a 1 and any string that begins with an even number of zeros.

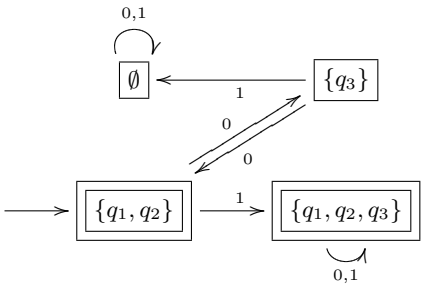

FIG. 6: The reduced DFA corresponding to N_{ex}.