Chapter 7

Equivalence Between DFAs and NFAs

We have seen that the NFAs have some capabilities that the DFAs lack; namely, to move to more than one state on an input symbol. We have also seen that in some cases this lead to a much simpler automaton to accept some language. The next question that we can ask is whether these capabilities of the NFAs increase their language-recognition power. In other words, does there exist a language that is accepted by some NFA, and that cannot be accepted by any DFA? The answer to that is - no. Nondeterminism does not increase the power of the automaton abstraction, and every language that is accepted by some NFA is also accepted by some DFA. We will formally prove this result in this chapter. This is the case also with many other computing models that we will consider in Theory of Computation. The features that we add to the models will normally not increase their power, but rather will make it easier and more natural to design the instances that accept certain languages.

Theorem 7.1: Equivalence Between DFAs and NFAs

The set of languages accepted by DFAs and the set of languages accepted by NFAs are the same. In other words, let D be the set

$$D = \{L | L \text{ is a language and } L \text{ is accepted by some DFA}\}$$

and let N be the set

$$N = \{L | L \text{ is a language and } L \text{ is accepted by some NFA}\}.$$

Then

$$D = N.$$

Note that the Theorem 7.1 claims the equality of two sets. As we can recall from Definition 1.2, equality of two sets A and B is most often proved by first proving that set A is a subset of set B, and then that set B is a subset of set A. If both of these are true, then it has to be that $A = B$. Therefore, the proof of the theorem will be divided into two parts.

Lemma 7.2: $D \subset N$

The set D of all languages accepted by DFAs is a subset of the set N of all languages accepted by NFAs ($D \subset N$). In other words, Every language accepted by some DFA is also accepted by some NFA.

Proof. This part of the proof looks obvious. Surely, if a language L is accepted by some DFA M, the same automaton can be seen as a special case of a NFA where each transition leads to only one state and there are no transitions that lead to an empty set of states. Therefore, M can also be seen as a NFA, and we have found an NFA that accepts the same language L. Therefore, for any language L accepted by some DFA, there is also a NFA that accepts that language. $\square$

Remark 7.1: DFAs Are Not Really NFAs

If we are to be fully strict, we would need to do a bit more work, because we cannot formally say that each DFA is also a NFA. Recall that the transition function for a DFA has a different range from the transition function for an NFA (one returns individual states and the other returns sets of states), therefore the transition function for some DFA cannot ever be the same as the transition function for some NFA. Since DFAs and NFAs are defined as 5-tuples, and the tuples are equal only if all of their components are equal, it follows that no DFA can be equal to some NFA. Therefore, to prove the Lemma 7.2 strictly formally, we would need to define a new NFA M' that would accept the same language as M, and then to prove that M and M' indeed accept the same language. This, however, is pretty trivial and unnecessarily cumbersome, so we will avoid it and just say (informally) that each DFA can be seen as an NFA.

Lemma 7.3: $N \subset D$

The set N of all languages accepted by NFAs is a subset of the set D of all languages accepted by NFAs ($N \subset D$). In other words, Every language accepted by some NFA is also accepted by some DFA.

Proof. This is a much more interesting part. Here we need to prove that, if we take an arbitrary language L that is accepted by some NFA M', we can find a DFA M that will accept the same language L. Since NFAs have more 'features' than DFAs, it is not at all obvious that such a DFA exists. Therefore, for a given language L over alphabet Σ and a given NFA M' that accepts that language, we need somehow to cunstrct the DFA M that will accept exactly the same language as M'.

So, given a language L accepted by some NFA $M' = (Q', \Sigma, \delta', q_0', F')$, how can we construct a DFA M that will accept the same language? We need to find all 5 components of the DFA M: the set of states, input alphabet, transition function, starting state and the set of accepting states. The input alphabet is the easiest part. Obviously, M and M' need to operate on the strings over the same input alphabet, so the input alphabet for M will also be Σ. The other components are far from obvious.

Let us start with the set of states, as that will reveal the basic principle behind constructing the DFA M. We know that at each point of the processing of any input string, M' will be in some set of states. We also know that this set of states, together with the remaining unprocessed symbols of the input string, is all that we need to know in order to complete the processing, as the subsequent states that M' goes through depend only on the current set of states and the remainder of the input string. Therefore, if we could capture this set of states in which M' is into a single state of the DFA M, we would almost solve our problem. Recall also that we are free to choose the set of states Q of our DFA M in any way we want. So far, the sets of states that we had were set of labels, such as $\{q_0, q_1, q_2, q_3\}$ in the case of the NFA in Example 6.2 or $\{0, 50, 100, 150, 200\}$ in the case of the Parking Ticket Machine DFA in Example 5.1. But nothing is preventing us from choosing this set to be anything we want - even a set of sets of labels! And this is exactly what we are going to use. The set of states of our DFA M will be the set of all *subsets of states* of the NFA M'! In other words, the set of states of M is a power set of the set Q'. Let us first see how this works on an example.

Example 7.1: Sets of States

Let a set of states of some NFA M' be $Q' = \{q_0, q_1, q_2\}$. The set (denoted by Q) of all subsets of the set of states Q' is

$$Q = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}.$$

Note that each element of the set Q is itself a set of states.

Remark 7.2: Notation for the States of M

To make things easier to understand and perhaps less confusing, we will denote by $[p_1', p_2', \ldots, p_m']$ the state of a DFA M that is a set of states $\{p_1', p_2', \ldots, p_m'\}$ of a NFA M'. This will make it easier to distinguish between a single state of M and the set of states of M'.

Note again that each individual element of our set Q of states of the DFA M itself is a set. So, a single state in Q is some (possibly empty) set of states from Q' of the NFA M'. For this construction to make sense, we need to ensure that the state in which M is after processing some string corresponds exactly to the set of states in which M' is after processing the same string. For example, M being in some state $[q_0, q_2]$ after processing the string w should mean that M' is in states q_0 and q_2 after processing the same string w. This is achieved by the suitable construction of the transition function δ of the DFA M.

What should our transition function δ be? We need to calculate, for each state r from Q and each input symbol a from Σ, the state from Q to which M moves from r on input symbol a. Since r is some set of states $[p'_0, p'_1, \dots, p'_n]$ from Q', let us consider what happens when M' is in this set of states, and it reads the input symbol a[1]. The set of states to which M' moves is $\cup_{i=1}^{n} \delta'(p_i, a)$. That is, we follow the transition from each state p_i on input symbol a ($\delta'(p_i, a)$). Each $\delta'(p_i, a)$ is a set of states. We then take the union of all these sets, which gives us the set of states to which M' moves on input symbol a. Therefore, we will define the transition δ for M from the state $r = \{p_1, p_2, \dots, p_n\}$ on input symbol a as

$$\delta(r, a) = \delta([p'_1, p'_2, \dots, p'_n], a) = \bigcup_{i=1}^{n} \delta'(p'_i, a).$$

In this way, we can define a transition for any set of states from Q' on any input symbol from Σ. We also ensure that M is, after reading some string w in the state that corresponds to the set of states in which M' is after reading the same string w.

Now that we have defined our set of states Q, input alphabet Σ and the transition function δ for our DFA M, the rest is relatively easy. The starting state of M needs to correspond to the starting state of M', and since the starting state of M' is q'_o, the starting state of M is $[q'_0]$. Finally, what should be the set of accepting states of M? We know that M' accepts a string if, after reading it completely, at least one of the states in which it ends is an accepting state. Therefore, each set of states from Q' that contains at least one accepting state will be an accepting state of Q. If F' is the set of accepting states of M', then we can formally define the set F of accepting states of Q as

$$F = \{r | r \in Q, r \cap F' \neq \emptyset\}.$$

This can be read as "F is the set of all states r, such that r is a state from Q and r has at least one element in common with F'". "r has at least one element in common with F'" means that at least one state from the set of states r is an accepting state.

We now have all the components of our DFA M. To recap, given a NFA $M' = (Q', \Sigma, \delta', q'_0, F')$ that accepts the language L, we can define an DFA $M = (Q, \Sigma, \delta, \{q_0\}, F)$ such that

- $Q = \mathcal{P}(Q')$
- For each $r = [p'_1, p'_2, \dots, p'_n]$, where each $p'_i \in Q'$, and each input symbol $a \in \Sigma$,

$$\delta(r, a) = \delta([p_1, p_2, \dots, p_n], a) = \bigcup_{i=1}^{n} \delta'(p_i, a).$$

- $F = \{r | r \in Q, r \cap F' \neq \emptyset\}$.

DFA M will then accept the same language L as the NFA M'.

Or will it? How do we actually know that M' would accept exactly the same language as M? Just because it seems "obvious" to us that this is the case, because M' was constructed in such a way that its operation mimics precisely the operation of M on the same string, it still doesn't mean it is necessarily true. To be fully formal and precise, we would need to actually prove this. We would need to prove that $L(M) = L(M')$. This would, again, be proven by proving first that $L(M) \subset L(M')$, and then that $L(M') \subset L(M)$. There is nothing particularly hard in this proof, and it is quite instructive of the proofs that two automata accept the same language. At the same time, it adds a bit of a burden to a very elegant construction that we have just seen. For this reason, we will give this proof in the Appendix A for the interested reader. $\square$

The construction in this proof is very instructive and commonly encountered in Theory of Computation, and it is called a *subset construction*.

In Lemma 7.2, we have proven that evey language accepted by a DFA is also accepted by some NFA. Conversely, in Lemma 7.3, we have proven that every language accepted by an NFA is also accepted by some

[1]Similar consideration was made when we introduced the transition function for strings for NFAs

DFA. These two results combined show that the set of languages accepted by DFAs is exactly the same as the set of languages accepted by NFAs.

Video 7.1: Example of a Subset Construction for NFAs

Design an NFA that accepts the language L of all strings over the alphabet $\{0, 1\}$ that start with 00 or end with 11 (or both).

We will finish this chapter with a few remarks. The DFA M that we constructed in our proof contains 2^n states, if n is the number of states of NFA M'. However, most of the time, most of these states are going to be unreachable from the start state $[q'_0]$ on any sequence of input symbols. There is no need to include these states in the final DFA, as they don't contribute to anything. Therefore, we can reduce the number of states in M by only considering the states that are reachable from the starting state. In this way, most often we would end up with a DFA that has roughly the same number of states as the corresponding NFA, but likely with more transitions. However, as we have seen in Example 6.1, there are also cases where, for a NFA M' with n states, any DFA that accepts the same language contains approximately 2^n states. In such cases, almost all of the states in our construction would be reachable.