Lecture 3

Nondeterministic finite automata

This lecture is focused on the *nondeterministic finite automata* (NFA) model and its relationship to the DFA model.

Nondeterminism is an important concept in the theory of computing. It refers to the possibility of having multiple choices for what can happen at various points in a computation. We then consider all of the possible outcomes that these choices can have, usually focusing on whether or not a sequence of choices *exists* that leads to acceptance. This may sound like a fantasy mode of computation not likely to be relevant from a practical viewpoint, because real computers don't make nondeterministic choices: each step a computer makes is uniquely determined by its configuration at any given moment. Our interest in nondeterminism is, however, not meant to suggest otherwise. Throughout this course we will see that nondeterminism is a powerful analytic tool (in the sense that it helps us to design things and prove facts), and its close connection with proofs and verification has fundamental importance.

3.1 Nondeterministic finite automata basics

Let us begin our discussion of the NFA model with its definition. The definition is similar to the definition of the DFA model, but with a key difference.

Definition 3.1. A *nondeterministic finite automaton* (or *NFA*, for short) is a 5-tuple

$$N = (Q, \Sigma, \delta, q_0, F), \tag{3.1}$$

where Q is a finite and nonempty set of *states*, Σ is an *alphabet*, δ is a *transition function* having the form

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q), \tag{3.2}$$

$q_0 \in Q$ is a *start state*, and $F \subseteq Q$ is a subset of *accept states*.

The key difference between this definition and the analogous definition for DFAs is that the transition function has a different form. For a DFA we had that $\delta(q, \sigma)$ was a *state*, for any choice of a state q and a symbol σ, representing the next state that the DFA would move to if it was in the state q and read the symbol σ. For an NFA, each $\delta(q, \sigma)$ is not a state, but rather a *subset of states*, which is equivalent to $\delta(q, \sigma)$ being an element of the power set $\mathcal{P}(Q)$. This subset represents all of the *possible states* that the NFA could move to when in state q and reading symbol σ. There could be just a single state in this subset, or there could be multiple states, or there might even be no states at all—it is possible to have $\delta(q, \sigma) = \varnothing$.

We also have that the transition function of an NFA is not only defined for every pair $(q, \sigma) \in Q \times \Sigma$, but also for every pair (q, ε). Here, as always in this course, ε denotes the empty string. By defining δ for such pairs we are allowing for so-called *ε-transitions*, where an NFA may move from one state to another without reading a symbol from the input.

State diagrams

Similar to DFAs, we sometimes represent NFAs with state diagrams. This time, for each state q and each symbol σ, there may be multiple arrows leading out of the circle representing the state q labeled by σ, which tells us which states are contained in $\delta(q, \sigma)$, or there may be no arrows like this when $\delta(q, \sigma) = \varnothing$. We may also label arrows by ε, which indicates where the ε-transitions lead. Figure 3.1 gives an example of a state diagram for an NFA. In this case, we see that $Q = \{q_0, q_1, q_2, q_3\}$,

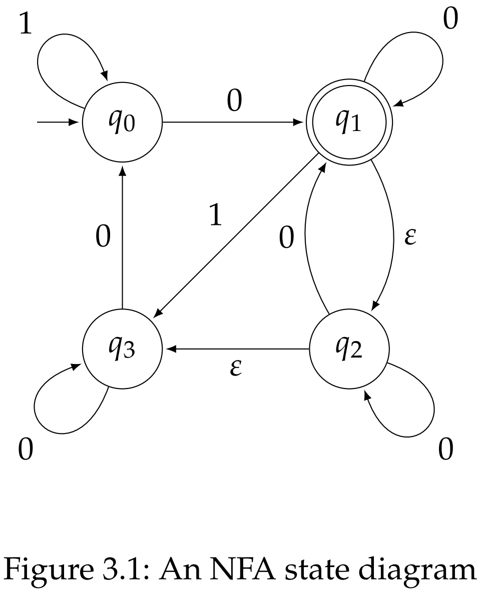


Figure 3.1: An NFA state diagram

$\Sigma = \{0,1\}$, q_0 is the start state, and $F = \{q_1\}$, just like we would have if this diagram were representing a DFA. The transition function, which must take the form

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q) \tag{3.3}$$

as the definition states, is given by

$$\begin{array}{lll} \delta(q_0,0) = \{q_1\}, & \delta(q_0,1) = \{q_0\}, & \delta(q_0,\varepsilon) = \varnothing, \\ \delta(q_1,0) = \{q_1\}, & \delta(q_1,1) = \{q_3\}, & \delta(q_1,\varepsilon) = \{q_2\}, \\ \delta(q_2,0) = \{q_1,q_2\}, & \delta(q_2,1) = \varnothing, & \delta(q_2,\varepsilon) = \{q_3\}, \\ \delta(q_3,0) = \{q_0,q_3\}, & \delta(q_3,1) = \varnothing, & \delta(q_3,\varepsilon) = \varnothing. \end{array} \tag{3.4}$$

NFA computations

Next let us consider the definition of acceptance and rejection for NFAs. This time we'll start with the formal definition and then try to understand what it says.

Definition 3.2. Let $N = (Q,\Sigma,\delta,q_0,F)$ be an NFA and let $w \in \Sigma^*$ be a string. The NFA N *accepts* w if there exists a natural number $m \in \mathbb{N}$, a sequence of states $r_0,\ldots,r_m$, and a sequence of either symbols or empty strings $\sigma_1,\ldots,\sigma_m \in \Sigma \cup \{\varepsilon\}$ such that the following statements all hold:

1. $r_0 = q_0$.
2. $r_m \in F$.
3. $w = \sigma_1 \cdots \sigma_m$.
4. $r_{k+1} \in \delta(r_k,\sigma_{k+1})$ for every $k \in \{0,\ldots,m-1\}$.

If N does not accept w, then we say that N *rejects* w.

As you may already know, we can think of the computation of an NFA N on an input string w as being like a single-player game, where the goal is to start on the start state, make moves from one state to another, and end up on an accept state. If you want to move from a state q to a state p, there are two possible ways to do this: you can move from q to p by reading a symbol σ from the input provided that $p \in \delta(q,\sigma)$, or you can move from q to p without reading a symbol provided that $p \in \delta(q,\varepsilon)$ (i.e., there is an ε-transition from q to p). In order to win the game, you must not only end on an accept state, but you must also have read every symbol from the input string w. To say that N accepts w means that *it is possible* to win the corresponding game.

Definition 3.2 essentially formalizes the notion of winning the game we just discussed: the natural number m represents the number of moves you make and

$r_0, \ldots, r_m$ represent the states that are visited. In order to win the game you have to start on state q_0 and end on an accept state, which is why the definition requires $r_0 = q_0$ and $r_m \in F$, and it must also be that every symbol of the input is read by the end of the game, which is why the definition requires $w = \sigma_1 \cdots \sigma_m$. The condition $r_{k+1} \in \delta(r_k, \sigma_{k+1})$ for every $k \in \{0, \ldots, m-1\}$ corresponds to every move being a legal move in which a valid transition is followed.

We should take a moment to note how the definition works when $m = 0$. Of course, the natural numbers (as we have defined them) include 0, so there is nothing that prevents us from considering $m = 0$ as one way that a string might potentially be accepted. If we begin with the choice $m = 0$, then we must consider the existence of a sequence of states $r_0, \ldots, r_0$ and a sequence of symbols of empty strings $\sigma_1, \ldots, \sigma_0 \in \Sigma \cup \{\varepsilon\}$, and whether or not these sequences satisfy the four requirements listed in the definition. There is nothing wrong with a sequence of states having the form $r_0, \ldots, r_0$, by which we really just mean the sequence r_0 having a single element. The sequence $\sigma_1, \ldots, \sigma_0 \in \Sigma \cup \{\varepsilon\}$, on the other hand, looks like it doesn't make any sense. It does, however, actually make sense: it is simply an *empty* sequence having no elements in it. The sensible way to interpret the condition $w = \sigma_1 \cdots \sigma_0$ in this case, which is a concatenation of an empty sequence of symbols or empty strings, is that it means $w = \varepsilon$.[1] Asking that the condition $r_{k+1} \in \delta(r_k, \sigma_{k+1})$ should hold for every $k \in \{0, \ldots, m-1\}$ is a vacuous statement, and therefore trivially true, because there are no values of k to worry about.

Thus, if it is the case that the initial state q_0 of the NFA we are considering happens to be an accept state and our input is the empty string, then the NFA accepts—for we can take $m = 0$ and $r_0 = q_0$, and the definition is satisfied. (It is worth mentioning that we could have done something similar in our definition for when a DFA accepts: if we allowed $n = 0$ in the second statement of that definition, it would be equivalent to the first statement, and so we really didn't need to take the two possibilities separately.)

Along similar lines to what we did for DFAs, we can define an extended version of the transition function of an NFA. In particular, if

$$\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q) \tag{3.5}$$

is a transition of an NFA, we define a new function

$$\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q) \tag{3.6}$$

[1] Note that it is a *convention*, and not something you can deduce, that the concatenation of an empty sequence of symbols gives you the empty string. It is similar to the convention that the sum of an empty sequence of numbers is 0 and the product of an empty sequence of numbers is 1.

as follows. First, we define the *ε-closure* of any set $R \subseteq Q$ as

$$\varepsilon(R) = \left\{ q \in Q \;:\; \begin{array}{l} q \text{ is reachable from some } r \in R \text{ by following} \\ \text{zero or more } \varepsilon\text{-transitions} \end{array} \right\}. \tag{3.7}$$

Another way of defining $\varepsilon(R)$ is to say that it is the intersection of all subsets $T \subseteq Q$ satisfying these conditions:

1. $R \subseteq T$.
2. $\delta(q, \varepsilon) \subseteq T$ for every $q \in T$.

We can interpret this alternative definition as saying that $\varepsilon(R)$ is the *smallest* subset of Q that contains R and is such that you can never get out of this set by following an ε-transition. With the notion of the ε-closure in hand, we define δ^* recursively as follows:

1. $\delta^*(q, \varepsilon) = \varepsilon(\{q\})$ for every $q \in Q$, and
2. $\delta^*(q, w\sigma) = \varepsilon\Bigl(\bigcup_{r \in \delta^*(q,w)} \delta(r, \sigma)\Bigr)$ for every $q \in Q$, $\sigma \in \Sigma$, and $w \in \Sigma^*$.

Intuitively speaking, $\delta^*(q, w)$ is the set of all states that you could potentially reach by starting on the state q, reading w, and making as many ε-transitions along the way as you like. To say that an NFA $N = (Q, \Sigma, \delta, q_0, F)$ accepts a string $w \in \Sigma^*$ is equivalent to the condition that $\delta^*(q_0, w) \cap F \neq \varnothing$.

Also similar to DFAs, the notation $\mathrm{L}(N)$ denotes the language *recognized* by an NFA N:

$$\mathrm{L}(N) = \bigl\{ w \in \Sigma^* \;:\; N \text{ accepts } w \bigr\}. \tag{3.8}$$

3.2 Equivalence of NFAs and DFAs

It seems like NFAs might potentially be more powerful than DFAs because NFAs have the option to use nondeterminism. Perhaps you already know from a previous course, however, that this is not the case, as the following theorem states.

Theorem 3.3. *Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a language. The language A is regular (i.e., recognized by a DFA) if and only if $A = \mathrm{L}(N)$ for some NFA N.*

Let us begin by breaking this theorem down, to see what needs to be shown in order to prove it. First, it is an "if and only if" statement, so there are two things to prove:

1. If A is regular, then $A = \mathrm{L}(N)$ for some NFA N.

2. If $A = \mathrm{L}(N)$ for some NFA N, then A is regular.

If you were in a hurry and had to choose one of these two statements to prove, you would likely choose the first—it's the easier of the two by far. In particular, suppose A is regular, so by definition there exists a DFA $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes A. The goal is to define an NFA N that also recognizes A. This is simple, we can just take N to be the NFA whose state diagram is the same as the state diagram for M. At a formal level, N isn't *exactly* the same as M; because N is an NFA, its transition function will have a different form from a DFA transition function, but in this case the difference is only cosmetic. More formally speaking, we can define $N = (Q, \Sigma, \mu, q_0, F)$ where the transition function $\mu : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$ is defined as

$$\mu(q, \sigma) = \{\delta(q, \sigma)\} \quad \text{and} \quad \mu(q, \varepsilon) = \varnothing \tag{3.9}$$

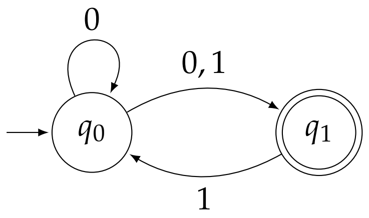
for all $q \in Q$ and $\sigma \in \Sigma$. It holds that $\mathrm{L}(N) = \mathrm{L}(M) = A$, and so we're done.

Now let us consider the second statement listed above. We assume $A = \mathrm{L}(N)$ for some NFA $N = (Q, \Sigma, \delta, q_0, F)$, and our goal is to show that A is regular. That is, we must prove that there exists a DFA M such that $\mathrm{L}(M) = A$. The most direct way to do this is to argue that, by using the description of N, we are able to come up with an *equivalent* DFA M. That is, if we can show how an arbitrary NFA N can be used to define a DFA M such that $\mathrm{L}(M) = \mathrm{L}(N)$, then we'll be done.

We will use the description of an NFA N to define an equivalent DFA M using a simple idea: each *state* of M will keep track of a *subset of states* of N. After reading any part of its input string, there will always be some subset of states that N could possibly be in, and we will design M so that after reading the same part of its input string it will be in a state corresponding to this subset of states of N.

A simple example

Let us see how this works for a simple example before we describe it in general. Consider the following NFA N:



If we describe this NFA formally, according to the definition of NFAs, it is given by

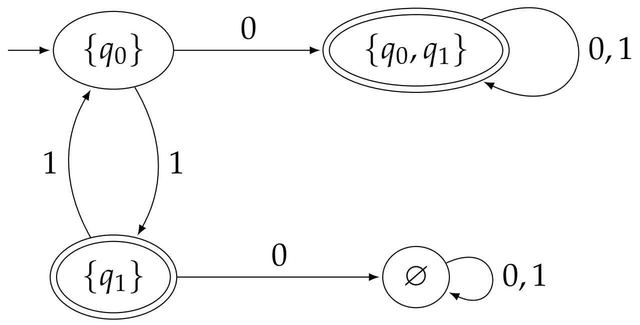
$$N = (Q, \Sigma, \delta, q_0, F) \tag{3.10}$$

where $Q = \{q_0, q_1\}$, $\Sigma = \{0,1\}$, $F = \{q_1\}$, and $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ is defined as follows:

$$\begin{aligned} &\delta(q_0,0) = \{q_0,q_1\}, && \delta(q_0,1) = \{q_1\}, && \delta(q_0,\varepsilon) = \varnothing, \\ &\delta(q_1,0) = \varnothing, && \delta(q_1,1) = \{q_0\}, && \delta(q_1,\varepsilon) = \varnothing. \end{aligned} \tag{3.11}$$

We are going to define an NFA M having one state for every subset of states of N. We can name the states of M however we like, so we may as well name them directly with the subsets of Q. In other words, the state set of M will be the power set $\mathcal{P}(Q)$.

Have a look at the following state diagram and think about if it makes sense as a good choice for M:



Formally speaking, this DFA is given by

$$M = (\mathcal{P}(Q), \Sigma, \mu, \{q_0\}, \{\{q_1\}, \{q_0, q_1\}\}), \tag{3.12}$$

where the transition function $\mu : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$ is defined as

$$\begin{aligned} &\mu(\{q_0\},0) = \{q_0,q_1\}, && \mu(\{q_0\},1) = \{q_1\}, \\ &\mu(\{q_1\},0) = \varnothing, && \mu(\{q_1\},1) = \{q_0\}, \\ &\mu(\{q_0,q_1\},0) = \{q_0,q_1\}, && \mu(\{q_0,q_1\},1) = \{q_0,q_1\}, \\ &\mu(\varnothing,0) = \varnothing, && \mu(\varnothing,1) = \varnothing. \end{aligned} \tag{3.13}$$

One can verify that this DFA description indeed makes sense, one transition at a time.

For instance, suppose at some point in time N is in the state q_0. If a 0 is read, it is possible to either follow the self-loop and remain on state q_0 or follow the other transition and end on q_1. This is why there is a transition labeled 0 from the state $\{q_0\}$ to the state $\{q_0, q_1\}$ in M—the state $\{q_0, q_1\}$ in M is representing the fact that N could be either in the state q_0 or the state q_1. On the other hand, if N is in the state q_1 and a 0 is read, there are no possible transitions to follow, and this is why

M has a transition labeled 0 from the state $\{q_1\}$ to the state $\varnothing$. The state $\varnothing$ in M is representing the fact that there aren't any states that N could possibly be in (which is sensible because N is an NFA). The self-loop on the state $\varnothing$ in M labeled by 0 and 1 represents the fact that if N cannot be in any states at a given moment, and a symbol is read, there still aren't any states it could be in. You can go through the other transitions and verify that they work in a similar way.

There is also the issue of which state is chosen as the start state of M and which states are accept states. This part is simple: we let the start state of M correspond to the states of N we could possibly be in without reading any symbols at all, which is $\{q_0\}$ in our example, and we let the accept states of M be those states corresponding to any subset of states of N that includes at least one element of F.

The construction in general

Now let us think about the idea suggested above in greater generality. That is, we will specify a DFA M satisfying $\mathrm{L}(M) = \mathrm{L}(N)$ for an *arbitrary* NFA

$$N = (Q, \Sigma, \delta, q_0, F). \tag{3.14}$$

One thing to keep in mind as we do this is that N could have ε-transitions, whereas our simple example did not. It will, however, be easy to deal with ε-transitions by referring to the notion of the *ε-closure* that we discussed earlier. Another thing to keep in mind is that N really is arbitrary—maybe it has 1,000,000 states or more. It is therefore hopeless for us to describe what's going on using state diagrams, so we'll do everything abstractly.

First, we know what the state set of M should be based on the discussion above: the power set $\mathcal{P}(Q)$ of Q. Of course the alphabet is Σ because it has to be the same as the alphabet of N. The transition function of M should therefore take the form

$$\mu : \mathcal{P}(Q) \times \Sigma \rightarrow \mathcal{P}(Q) \tag{3.15}$$

in order to be consistent with these choices. In order to define the transition function μ precisely, we must therefore specify the output subset

$$\mu(R, \sigma) \subseteq Q \tag{3.16}$$

for every subset $R \subseteq Q$. One way to do this is as follows:

$$\mu(R, \sigma) = \bigcup_{q \in R} \varepsilon(\delta(q, \sigma)). \tag{3.17}$$

In words, the right-hand side of (3.17) represents every state in N that you can get to by (i) starting at any state in R, then (ii) following a transition labeled σ, and finally (iii) following any number of ε-transitions.

The last thing we need to do is to define the initial state and the accept states of M. The initial state is $\varepsilon(\{q_0\})$, which is every state you can reach from q_0 by just following ε-transitions, while the accept states are those subsets of Q containing at least one accept state of N. If we write $G \subseteq \mathcal{P}(Q)$ to denote the set of accept states of M, then we may define this set as

$$G = \big\{R \in \mathcal{P}(Q) \,:\, R \cap F \neq \varnothing\big\}. \tag{3.18}$$

The DFA M can now be specified formally as

$$M = (\mathcal{P}(Q), \Sigma, \mu, \varepsilon(\{q_0\}), G). \tag{3.19}$$

Now, if we are being honest with ourselves, we cannot say that we have *proved* that for every NFA N there is an equivalent DFA M satisfying $\mathrm{L}(M) = \mathrm{L}(N)$. All we've done is to define a DFA M from a given DFA N that *seems* like it should satisfy this equality. We won't go through a formal proof that it really is the case that $\mathrm{L}(M) = \mathrm{L}(N)$, but it is worthwhile to think about how we would do this if we had to. (It is, in fact, true that $\mathrm{L}(M) = \mathrm{L}(N)$, so you shouldn't worry that we're trying to prove something that might be false.)

First, if we are to prove that the two languages $\mathrm{L}(M)$ and $\mathrm{L}(N)$ are equal, the natural way to do it is to split it into two statements: $\mathrm{L}(M) \subseteq \mathrm{L}(N)$ and $\mathrm{L}(N) \subseteq \mathrm{L}(M)$. This is often the way to prove the equality of two sets. Nothing tells us that the two statements need to be proved in the same way, and by doing them separately we give ourselves more options about how to approach the proof. Let's start with the subset relation $\mathrm{L}(N) \subseteq \mathrm{L}(M)$, which is equivalent to saying that if $w \in \mathrm{L}(N)$, then $w \in \mathrm{L}(M)$. We can now fall back on the definition of what it means for N to accept a string w, and try to conclude that M must also accept w. It's a bit tedious to write everything down carefully, but it is possible and maybe you can convince yourself that this is so. The other relation $\mathrm{L}(M) \subseteq \mathrm{L}(N)$ is equivalent to saying that if $w \in \mathrm{L}(M)$, then $w \in \mathrm{L}(N)$. The basic idea here is similar in spirit, although the specifics are a bit different. This time we start with the definition of acceptance for a DFA, applied to M, and then try to reason that N must accept w.

A different way to prove that the construction works correctly is to make use of the functions δ^* and μ^*, which are defined from δ and μ as we discussed in the previous lecture and earlier in this lecture. In particular, using induction on the length of w, it can be proved that

$$\mu^*(\varepsilon(R), w) = \bigcup_{q \in R} \delta^*(q, w) \tag{3.20}$$

for every string $w \in \Sigma^*$ and every subset $R \subseteq Q$. Once we have this, we see that $\mu^*(\varepsilon(\{q_0\}), w)$ is contained in G if and only if $\delta^*(q_0, w) \cap F \neq \varnothing$, which is equivalent to $w \in \mathrm{L}(M)$ if and only if $w \in \mathrm{L}(N)$.

In any case, I do not ask that you try to verify one of these proofs—but only that you *think about* how it would be done.

On the process of converting NFAs to DFAs

It is a very typical type of exercise in courses such as CS 360 that students are presented with an NFA and asked to come up with an equivalent DFA using the construction described above. I will not ask you to perform such a mindless tasks as this. Indeed, it will be helpful later in the course for us to observe that the construction itself can be implemented by a computer, and therefore requires absolutely no creativity whatsoever.

My primary aim when teaching CS 360 is to introduce you to theoretical computer science, and memorizing a simple technique like the one we just saw is not what theoretical computer science is all about. Theoretical computer science is about coming up with the proof in the first place, not behaving like a computer by performing the technique over and over. In this case it was all done in the 1950s by Michael Rabin and Dana Scott.

Instead of asking you to implement simple constructions, such as converting NFAs into DFAs, I am likely to ask you to solve problems that require creativity and reasoning about new ideas that you haven't seen yet. For example, I may tell you that A is a regular language, that B is obtained from A in a certain way, and ask you to prove that B is regular. Now that you know that NFAs and DFAs are equivalent, you can use this fact to your advantage: if you want to prove that B is regular, it suffices for you to give an NFA N satisfying $\mathrm{L}(N) = B$. Maybe it is possible to do this starting from the assumption that A is regular, and therefore recognized by a DFA.

If, for some reason, you find that you really do want to take a given NFA N and construct a DFA M recognizing the same language as N, it is worth pointing out that you really don't need to write down every subset of states of N and then draw the arrows. There will be exponentially many more states in M than in N, and it will often be that many of these states are useless, possibly unreachable from the start state of M. A better option is to first write down the start state of M, which corresponds to the ε-closure of the set containing just the start state of N, and only draw new states of M as you need them.

In the worst case, however, you might actually need all of those states. There are examples known of languages that have an NFA with n states, while the smallest DFA for the same language has 2^n states, for every choice of a positive integer n. So, while NFAs and DFAs are equivalent in computational power, there is sometimes a significant cost to be paid in converting an NFA into a DFA, which is that this might require the DFA to have a huge number of states.