Chapter 6

Non-Deterministic Finite Automata

The first generalisation of the concept of a deterministic finite automaton that we are going to consider is a *non-deterministic finite automaton (NFA)*. Recall that a DFA can be in only one state at a time. This also means that it can only move to one state from a given state on a given input symbol. This is why we defined the transition function of a DFA to have a domain $Q \times \Sigma$ and the range Q. That is, for each state q and each input symbol $a \in \Sigma$, it returns a single element of the set of states Q. NFAs will relax this assumption. A NFA will potentially be in *more than one* state at a time. Another assumption that NFAs relax is that there has to be a transition to some state for each state and each input symbol. If we look at a representation of a DFA using a transition graph, this assumption meant that there has to exist an arrow from each state on each input symbol. This will not be the case for NFAs.

Before formally defining an NFA, let us consider the following example.

Example 6.1: Bad Case for a DFA

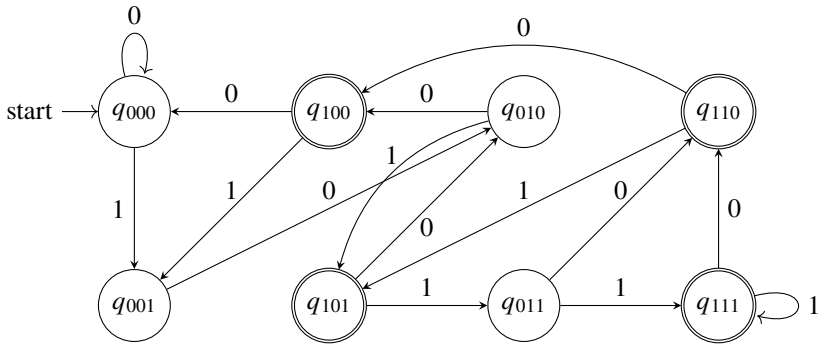
Design a DFA that accepts the language of all strings over alphabet $\Sigma = \{0, 1\}$ whose third symbol from the right is 1.

$$L = \{x1ab | x \in \Sigma^*, a \in \Sigma, b \in \Sigma\}.$$

How would we go about creating this DFA? As usual, the first thing we do is to determine the set of states the DFA will have, which is, in turn, determined by the information we want our DFA to remember. At a first glance, it seems that the only information we need is whether the third symbol from the right of the string read so far is 0 or 1. So, as a first attempt, we might try designing a DFA with just two states - q_0 when the third symbol from the right in the string read so far is not 1 (either because it is 0 or because fewer that three symbols have been read), and q_1 if the third symbol from the right in the string read so far is 1. The state q_1 should be the only accepting state (for the obvious reasons). However, this approach quickly gets us into trouble. What would transitions from each of the states be? If, for example, we are in the state q_1 and we read a 0 as the new symbol, where should we transition? We don't know this, because if we use just these two states, we have no idea what the *second* symbol from the right in the previously read string was, and this symbol becomes the new third symbol from the right when another symbol is read. So, it seems that we also need to remember the second symbol from the right for the string read so far. If this symbol is 1, then reading any new symbol would make us go to the state q_1. Otherwise, if it was 0 or we haven't read two symbols yet, we should go to q_0. Following the same reasoning, we also need to remember the last symbol read, as this will become the new third symbol from the right if we read two more symbols. Remember that the states of a DFA cannot assume anything about the remaining symbols of the input string that are still to be read (including how many symbols are there).

So, it seems that we have concluded that we need to remember the last three symbols read. What are the possibilities for the last three symbols read? It is easy to see that there are 8 different possibilities: $000, 001, 010, 011, 100, 101, 110, 111$. This gives us the 8 possibilities to remember, corresponding to 8 states of the DFA. It is very easy to derive the transitions between the states now. For example, if we are in the state 000 and we read a symbol 1, we should move to the state 001, as this now represents the last 3 symbols read. It is also very easy to see what states of these should be the accepting states. These are, for obvious reasons, 100, 101, 110 and 111. Now, how about a situation where fewer than 3 symbols have been read? Do we need separate (obviously non-accepting) states $0, 1, 00, 01, 10$ and 11, corresponding to

different combinations when one or two symbols have been read so far? It turns out that we actually do not need states for these. For example, if we have read only one symbol and it is 0, what do we need to read afterwards in order to accept the string? We need to read 1, followed by any two symbols. But this is exactly the same situation as for the state 000! In that state, we will also move to an accepting state if we read 1, followed by any two symbols. Therefore, the state 000 also covers the situation where only 0 (or, indeed, 00) have been read so far, and also the situation where nothing has been read so far. Therefore, this will be our starting state. Similarly, the state 001 covers the situation where only one symbol, 1, has been read so far. In fact, it is easy to see that adding enough 0's at the start of our 1- and 2-symbol strings gives us exactly the state that corresponds to the situation when just that string has been read so far (e.g. 010 when only 10 has been read). Therefore, our DFA (when we name the state 000 as q_{000}, the state 001 as q_{001} and so on) is



This isn't too bad, but what would have happened if we wanted to design a DFA that accepts the strings where the 10-th symbol from the right is 1? We would need to remember 10 last symbols, which would give us $2^{10} = 1024$ states for different possibilities for the last 10 symbols! The transition graph for such a DFA would not be easy to draw, to say the least.

The question we can ask is - can we design a smaller DFA (in terms of the number of states) that will accept the language in Example 6.1? The answer is, sadly, no. Let us assume that we have managed to find a smaller DFA (say, DFA M) that accepts this language. Since it is smaller, it has to have fewer than 8 states. Since, as we said, there are 8 possible strings of length 3 (remembering that there are 8 possibilities for the last 3 symbols read), this means that M would end up in the same state when it reads two different strings of length 3[1]. Let us say that these are the strings 010 and 000. But what would happen from here if we read another symbol, say 0? If the last three symbols read before that were 010, the DFA should transit to an accepting state. But if the last three symbols read before that were 000, then it should transit to a non-accepting state. Therefore, on the input symbol 0, the DFA must transition to different states when 010 and 000 are the last three symbols read. This cannot happen if the DFA is in the same state when either of these are the three last symbols read, because we said that the state to which the DFA goes depends only on the current state of the DFA and the next symbol read! Therefore, it cannot happen that the DFA that accepts this language ends up in the same state when it reads 000 and 010. It is easy to see that the same is true for any other two combinations for the last three symbols read. This means that there cannot be fewer than 8 states in any DFA that accepts this language!

What we have shown so far is that there is a DFA that accepts the language in Example 6.1, but also that every DFA that accepts that language has to have at least 8 states. Similarly, it is easy to see that any DFA that accepts the language of the strings over alphabet $\{0, 1\}$ whose 10-th symbol from the right is 1 has to have at least 1024 states. It is needless to say that specifying such a DFA is very cumbersome and building them would be very painful.

The next question we can ask is - what is it that makes it a requirement for every DFA that accepts our langauge to be so large? It is the fact that we need to remember lots of information. We need to remember the last 3 symbols read, because we have to design the states in a way that does not make any assumption about the remaining symbols of the input string that are to be read. In particular, we cannot make an assumption that the currently read symbol is the third symbol from the right, nor that we have read the complete input string. For each state, we have to assume both that we might have read the complete input string (and, thus, we need to be in the correct state with respect to acceptance/non-acceptance of the string), but also that there

[1] This is the so-called *pigeonhole* principle. If you have m pigeonholes and n pigeons, where $n > m$, and if you try to fit all the pigeons in the pigeonholes, you will end up putting at least two pigeons in the same pigeonhole. This is because there are more pigeons than there are pigeonholes

might be an arbitrary number of symbols still to be read in the input string. This is because the DFA can be in only one state at a time. Our life would be much easier if our automaton could be in TWO states at the same time - one state where it assumes that the last symbol read is the third symbol from the right (and, thus, we just need to read the remaining two symbols and move to an accepting state), and another where it assumes that there are more than two symbols still to be read. And this is possible if we use a non-deterministic finite automaton.

6.1 Formal Definition of a Non-Deterministic Finite Automaton

Non-deterministic finite automaton (NFA) can be defined in a very similar way to a DFA.

Definition 6.1: Non-Deterministic Finite Automaton (NFA)

Non-Deterministic Finite Automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of *states*
- Σ is a finite *alphabet* (set of possible input symbols)
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is the transition function
- $q_0 \in Q$ is the *start state*
- $F \subset Q$ is the set of *accepting states*

We can note that this definition is almost the same as for a DFA, with one important difference, which is that the range of the transition function is now a power set of the set of states. This means that this function returns, for a state and a value, a *subset* of states of the automaton. There are two important implications of this. Firstly, since a NFA transitions, from a state on an input, to a set of states, the NFA can be in more than one state at a time. Secondly, since the empty set is an element of every power set, the transition function can also return, for some state and some input symbol, the empty set. This means that the NFA can also end up in *no state at all*. This can be interpreted as the NFA getting "stuck" in execution, being unable to proceed. We can summarise these different possibilities for the return value of a transition function. If, for some state p and some input symbol a, we have

- $\delta(p, a) = \{q\}$, then the NFA transitions to a single state q from state p on input symbol a; this is similar as for DFAs.
- $\delta(p, a) = \{q_1, q_2, \ldots, q_n\}$, then the NFA transitions to *all* of the states $q_1, q_2, \ldots, q_n$ from state p on input symbol a.
- $\delta(p, a) = \emptyset$, then the NFA transitions to no state at all from state p on input symbol a.

Remembering that an NFA can be in more than one state at a time, we can ask the question of what happens when an NFA is in some set of states and it reads some input symbol a. In that case, we need to follow transitions from all of the states the NFA is, on input symbol a. After the NFA ends up processing symbol a it will be in all of the states that are reached from any of the state it was previously in on input symbol a. So, what happens then if the NFA transitions to no states from some state p on input symbol a? If the NFA was *only* in state p, then the NFA would terminate its operation and rejects the input string. There is no way to proceed, as the NFA got stuck. Otherwise, we follow the transitions from the other states the NFA was in.

A convenient way to think of an operation of an NFA is to think that, whenever non-determinism is encountered, the NFA takes all possible options in parallel. That is, if the NFA transitions to two states from some state on some input symbol, two copies of the NFA are created and each of them transitions to one of these two states. Then we have two NFAs operating in parallel, each taking one path of operation. These paths can, of course, further branch into multiple paths when non-determinisim is encountered again. If in one of these paths we encountered a transition that takes the NFA in it into an empty set of states, this path is simply discarded, but the other paths continue in parallel.

One final thing to ask is when do we say that a string is accepted by an NFA. In the general case, an NFA will be in some set of states when it finishes reading the input string. Some of these might be accepting states and some might be non-accepting. We will say that a string is accepted if *any* of the states it ends up

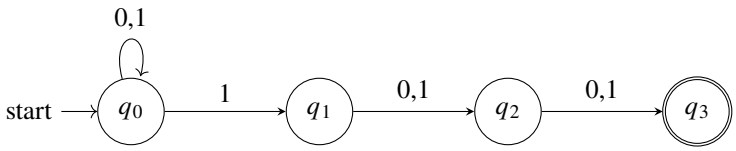
in after reading the whole string is an accepting state. It does not matter if it also ends up in some of the non-accepting states, as long as at least one state is accepting. If we again look at an operation of an NFA as taking different paths of operation when non-determinism is encountered, we can say that a string is accepted if at least one path of the NFA operation on that string ends up in an accepting state. It does not matter how many paths lead to non-accepting states, or to the NFA getting stuck, as long as at least one of paths leads to an accepting state.

Example 6.2: Revisitng the Bad Case for a DFA

Design a NFA that accepts the language of all strings over alphabet $\Sigma = \{0, 1\}$ whose third symbol from the right is 1.

$$L = \{x1ab | x \in \Sigma^*, a \in \Sigma, b \in \Sigma\}.$$

We can remember that the main problem for any DFA that accepts this language was that it had to remember the three last symbols of the string read so far, because it cannot know whether it has reached the end of the input string or not. With an NFA, we can address this problem by taking two different paths of execution when the symbol 1 is read - one where we assume that the 1 just read is the third symbol from the right (and thus ending up in an accepting state when two more symbols are read), and one where we assume that this is not the case. That is, each time we read 1 as an input symbol, we transition to two states, the first of these assuming that the input symbol is the third symbol from the right, and the other assuming it is not. An NFA that accepts our language L can be represented by a transition graph



The transition function for this NFA is given by the table

δ	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q2\}$	$\{q_2\}$
q_2	$\{q_3\}$	$\{q_3\}$
q_3	$\emptyset$	$\emptyset$

This certainly looks much more concise that the DFA that accepts the same language (see Example 6.1!

We can note a few interesting features of this automaton. The first is non-determinism in the state q_0 when the input symbol 1 is read, where the NFA transitions to both q_1 and back to q_0. Since the NFA also transitions from q_0 to q_0 on input 0, we can conclude that the NFA will always be in the state q_0. The state q_1 represents the state where the symbol 1 is read. This is the state in which we assume that the symbol 1 just read is the third symbol from the right. So, whatever symbol we read in this state, we move to q_2 and then, after one more symbol is read, to the accepting state q_3. Another interesting feature is that there are no transitions out of state q_3. This is because $\delta(q_3, 0) = \emptyset$ and $\delta(q_3, 1) = \emptyset$. Therefore, any path of operation where we end up in state q_3, and where another symbol is read after that, will lead to the NFA getting stuck and that path abandoned. This corresponds to the situation where our assumption that the symbol 1 read in the state q_0 was the third symbol from the right proves to be wrong.

At the end of processing of any string, our NFA will certainly be in state q_0 (because it is always in that state), and possibly in some other states. If one of these states is q_3, then we know that we have reached that state from q_0, reading first symbol 1 and then any two other symbols. So, the third symbol from the right is, indeed, 1 and the string is accepted. Otherwise, if q_3 is not one of the states the NFA is after it ends processing an input string, it means we couldn't have possibly read the symbol 1 followed by any two other symbols, because we would have then eneded in state q_3. Therefore, the third symbol from the right of an input string is not 1, and the input string is not accepted.

Video 6.1: Operation of an NFA

Demonstration of how the NFA from Example 6.2 operates on input string $w = 00100$.

6.2 Transition Function for NFAs and Language Accepted by a NFA

We are now ready to formally define acceptance of a string by an NFA. We already said, informally, that a string is accepted by an NFA if at least one of the states in which the NFA is when it processes the whole string is an accepting state. But to formalise this, we first need to extend the definition of the transition function from individual symbols to strings, similarly as we did with DFAs.

Definition 6.2: NFA Transition Function for Strings - Informal

For the NFA $M = (Q, \Sigma, \delta, q_0, F)$, transition function $\hat{\delta}$ for strings over Σ returns, for a given state q and string w, the *set of states* in which the automaton M is when it reads the whole string w, if it starts reading in the state q.

Definition 6.3: Transition Function for Strings - Formal

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a NFA. We define the transition function for strings over Σ, $\hat{\delta} : Q \times \Sigma \to \mathcal{P}(Q)$ in the following way:

1. If $w = \epsilon$, then $\hat{\delta}(q, w) = \{q\}$, for every state $q \in Q$.
2. If $w = xa$, where $x \in \Sigma^*$ and $a \in \Sigma$, then let $\hat{\delta}(q, x) = \{q_1, q_2, \ldots, q_k\}$. Then

$$\hat{\delta}(q, w) = \hat{\delta}(q, xa) = \bigcup_{i=1}^{k} \delta(q_i, a),$$

for every state $q \in Q$.

Let us analyse this definition in detail. It is, again, an inductive definition on the length of a string. The base case, for the empty string, is similar as for the DFAs. An NFA, if it is in the state q and reads an empty string, remains in the state q and this is the only state it is in. Note that the value that the function returns is a set that consists only of the element q. The second case, for strings of length at least 1, is a bit more complicated, but it follows precisely our intuitive notion of what happens in the operation of the NFA. To determine the set of states in which the NFA is after reading the whole string, if it starts in the state q, we first determine the set of states in which the NFA is after reading the whole string except for the last symbol a in it. This is $\hat{\delta}(q, x)$ and it is the set of states $\{q_1, q_2, \ldots, q_k\}$. Afterwards, we follow the transitions from each of the states from this set, on input symbol a. That is, we determine $\delta(q_1, a), \delta(q_2, a), \ldots, \delta(q_k, a)$. This gives us k sets of states. Finally, we take the union of all these sets to be the set of states in which the DFA is after reading the complete input string, including the last symbol a ($\bigcup_{i=1}^{k} \delta(q_i, a)$).

Remark 6.1: Ending up In Empty Set of States

We can ask what happens in the above definition if $\hat{\delta}(q, x) = \emptyset$. This does not present any problems, because in that case the set of states $\{q_1, q_2, \ldots, q_k\}$ in which the NFA ends up when processing the string x is simply the empty set. That is to say, $k = 0$. We then have that $\hat{\delta}(q, xa) = \bigcup_{i=1}^{0} \delta(q_i, a) = \emptyset$ (because there are no sets in this union). So, the Definition 6.3 is consistent with our intuitive notion of what happens when an NFA ends up in an empty set of states at some point when processing an input string - it will remain in the empty set of states (or, in practical terms, stuck) when processing all the remaining symbols of the string.

Video 6.2: Transition Function of an NFA

Demonstration of how to calculate the transition function of the NFA from Example 6.2 for a starting state q_0 and an input string $w = 00100$.

As in the case of DFAs, using the transition function for strings we can now define a language accepted by an NFA.

Definition 6.4: Language Accepted by an NFA

For a NFA $M = (Q, \Sigma, \delta, q_0, F)$, the language accepted by M (usually denoted by $L(M)$ or L_M) is the set of all strings w over Σ, such that $\hat{\delta}(q_0, w)$ contains at least one state from F.

$$L(M) = \{w | \hat{\delta}(q_0, w) \cap F \neq \emptyset\}.$$

$\hat{\delta}(q_0, w) \cap F \neq \emptyset$ is "at least one state from $\hat{\delta}(q_0, w)$ is an accepting state" expressed in the language of sets. The intersection of two sets not being the empty set means that there is at least one common element between these two sets. So, the set of the states in which M is after it finishes processing the string w if it starts in the start state q_0, $\hat{\delta}(q_0, w)$, and the set of accepting states F have at least one element in common, which means that at least one state from $\hat{\delta}(q_0, w)$ is an accepting state.