1 Nondeterministic Finite Automata

Non-deterministic Finite Automata, NFAs for short, are a generalization of the machines we have already defined, which are often called Deterministic Finite Automata by contrast, or DFAs for short. The reason for the name will become clear later.

As with a DFA, an NFA is simply a graph with edges labeled by single letters from the input alphabet $\Sigma$. There is one structural change.

For each vertex $v$ and each character $a \in \Sigma$, the number of edges exiting $v$ labeled with $a$ is unconstrained; it could be 0, 1, 2 or more edges.

This obliges us to redefine what an automaton $M$ is doing, given an input $x$. Quite simply, $M$ on input $x$ determines all the vertices that can be reached by paths starting at the start vertex, $\text{start}$, and labeled $x$. If any of these reachable vertices is in the set of Accepting or Final vertices then $M$ is defined to accept $x$. Another way of looking at this is that $M$ recognizes $x$ exactly if there is some path (and possibly more than one) labeled $x$ from $\text{start}$ to a vertex (state) $q \in F$; such a path is called a recognizing or accepting path.

Example 1 Let $A = \{w \mid$ the third to last character in $w$ is a “b”$\}$. The machine $M$ with $L(M) = A$ is shown in Figure 1.

Note that on input $abbb$ all four vertices can be reached, whereas on input $abab$ the second from rightmost vertex cannot be reached.

As we will see later, the collection of languages recognized by NFAs is exactly the collection of languages recognized by DFAs. That is, for each NFA $N$ there is a DFA $M$ with $L(M) = L(N)$. Thus NFAs do not provide additional computational power. However, they can be more compact and easier to understand, and as a result they are quite useful.

Indeed, the language from Example 1 is recognized by the DFA shown in Figure 2.

It is helpful to consider how one might implement an NFA. One way is to maintain a bit vector over the vertices, so as to record which vertices are reachable currently. That is, suppose that reading input $w$ can lead to any of the vertices in...
Figure 1: NFA recognizing \( \{ w \mid \text{the third to last character in } w \text{ is a } b \} \).

Figure 2: DFA recognizing \( \{ w \mid \text{the third to last character in } w \text{ is a } b \} \). By naming the last three characters or all the characters in the string, each vertex specifies the strings for which the DFA reaches it.
set \( R \), say. The set \( S \) of vertices reachable on reading a further character \( a \) is obtained by following all the edges labeled \( a \) that exit any of the vertices in \( R \). So in the automata of Figure 1, the set of vertices reachable on reading input \( b \) is \{“any string”, “last char read is a \( b \)”\}, and the set of vertices reachable on reading \( ba \) is \{“any string”, “next to last char read is a \( b \)”\}.

We define recognition for NFAs as follows: An NFA \( M \) recognizes input string \( w \) if there is a path from \( M \)'s start vertex to a final vertex (state) such that the labels along the edges of the path, when concatenated, form the string \( w \). That is, were we to follow this path, reading the edge labels as we go, the string read would be exactly \( w \); we say this path has label \( w \). The implementation described above checks every path with label \( w \); for acceptance, one needs to have at least one label \( w \) path leading to a final vertex. We call such a path a recognizing or accepting path for \( w \).

To define NFAs formally, we need to redefine the destination (transition) functions \( \delta \). Now \( \delta \) takes as its arguments a set of vertices (states) \( R \subseteq V \) and a character \( a \in \Sigma \) and produces as output another set of vertices (states) \( S \subseteq V \): \( \delta(R, a) = S \). The meaning is that the set of vertices (states) reachable from \( R \) on reading \( a \) is exactly the set of vertices (states) \( S \).

\( \delta^* \) is also redefined. \( \delta^*(R, w) = S \) means that the set of vertices (states) \( S \) are the possible destinations on reading \( w \) on starting from a vertex (state) in \( R \). So, in particular, \( w \) is recognized by \( M \), \( w \in L(M) \), exactly if \( \delta^*(\{\text{start}\}, w) \cap F \neq \Phi \); i.e., when starting at vertex \( \text{start} \), on reading \( w \), at least one destination vertex (state) is an accepting or final vertex (state).

\( \delta \) is often defined formally as follows: \( \delta : 2^Q \times \Sigma \rightarrow 2^Q \). In this context, \( 2^Q \) means the collection of all possible subsets of \( Q \), the set of states of the NFA, and so \( \delta \) does exactly what it should. Its inputs are (i) a set of vertices (states), that is one of the elements of the collection \( 2^Q \), and (ii) a character in \( \Sigma \); its output is also a set of vertices (states).

Next, we add one more option to NFAs: edges labeled with the empty string, as shown in Figure 3. We call these edges \( \lambda \)-edges for short.

![Figure 3: \( \lambda \) label on an edge.](image)

The meaning is that if vertex (state) \( p \) is reached vertex (state) \( q \) can also be reached without reading any more input. Again, the same languages as before are recognized by NFAs with \( \lambda \)-edges and by DFAs; however, the \( \lambda \)-edges are very convenient in creating understandable machines. The meaning of the destination functions \( \delta \) and \( \delta^* \) are unchanged.
Example 2 Let $A$ and $B$ be languages over the alphabet $\Sigma$ that are recognized by NFAs $M_A$ and $M_B$, respectively. Then the NFA shown in Figure 4 recognizes the language $A \cup B$.

Its first step, prior to reading any input, is to go to the start vertices for machines $M_A$ and $M_B$. Then the computation in these two machines is performed simultaneously. The set of final vertices for $M$ is the union of the final vertices for $M_A$ and $M_B$; thus $M$ reaches a final vertex on input $w$ exactly if at least one of $M_A$ or $M_B$ reaches a final vertex on input $w$. In other words $L(M) = L(M_A) \cup L(M_B)$.

**Deterministic vs. Nondeterministic** Another way of viewing the computation of an NFA $M$ on input $x$ is that the task is to find a path labeled $x$ from the start vertex to a final vertex if there is one, and this is done correctly by (inspired) guessing. This process of correct guessing is called Non-deterministic computation. Correspondingly, if there is no choice or uncertainty in the computation, it is said to be Deterministic. This is not an implementable view; it is just a convenient way to think about what nondeterminism does.

### 1.1 Closure Properties

**Definition 3** A language is said to be regular if it can be recognized by an NFA.
The regular languages obey three closure properties: if $A$ and $B$ are regular, then so are $A \cup B$, $A \circ B$ and $A^*$. We have already seen a demonstration of the first of these. We now give a simpler demonstration of the first property and then show the other two.

First, the following technical lemma is helpful.

**Lemma 4** Let $M$ be an NFA. Then there is another NFA, $N$, which has just one final vertex, and which recognizes the same language as $M$: $L(N) = L(M)$.

**Proof:** The idea is very simple: $N$ is a copy of $M$ with one new vertex, $\text{new\_final}$, added; $\text{new\_final}$ is $N$’s only final vertex. $\lambda$-edges are added from the vertices that were final vertices in $M$ to $\text{new\_final}$, as illustrated in Figure 5.

![Figure 5: NFA N with a single final vertex](image)

Recall that a recognizing path in an NFA goes from its start vertex to a final vertex. It is now easy to see that $M$ and $N$ recognize the same language. The argument has two parts.

First, any recognizing path in $M$, for string $w$ say, can be extended by a single $\lambda$-edge to reach $N$’s final vertex, and thereby becomes a recognizing path in $N$; in addition, the edge addition leaves the path label unchanged (as $w\lambda = w$). It follows that if $M$ recognizes string $w$ then so does $N$. In other words, $L(M) \subseteq L(N)$.

Second, removing the last edge from a recognizing path in $N$ yields a recognizing path in $M$ having the same path label (for the removed edge has label $\lambda$). It follows that if $N$ recognizes string $w$ then so does $M$. In other words, $L(N) \subseteq L(M)$.

Together, these two parts yield that $L(M) = L(N)$, as claimed.

We are now ready to demonstrate the closure properties.
If A and B are regular then so is A $\cup$ B  We show this using the NFA $M_{A\cup B}$ in Figure 6. $M_{A\cup B}$ is built using $M_A$ and $M_B$, NFAs with single final vertices recognizing $A$ and $B$, respectively. It has an additional start vertex $\text{start}$, connected to $M_A$ and $M_B$’s start vertices by $\lambda$-edges, and an additional final vertex $\text{final}$, to which $M_A$ and $M_B$’s final vertices are connected by $\lambda$-edges.

As in the earlier construction, each recognizing path in $M_{A\cup B}$ corresponds to a recognizing path in one of $M_A$ and $M_B$, and vice-versa, and thus $w \in L(M_{A\cup B})$ if and only if $w \in A \cup B$. We now show this more formally.

**Lemma 5** $M_{A\cup B}$ recognizes $A \cup B$.

**Proof:** We show that $A \cup B = L(M_{A\cup B})$ by showing that $L(M_{A\cup B}) \subseteq A \cup B$ and that $A \cup B \subseteq L(M_{A\cup B})$.

To show that $L(M_{A\cup B}) \subseteq A \cup B$, it is enough to show that if $w \in L(M_{A\cup B})$ then $w \in A \cup B$ also. So let $w \in L(M_{A\cup B})$ and let $P$ be a recognizing path for $w$ in $M_{A\cup B}$. Removing the first and last edges of $P$, which both have label $\lambda$, yields a recognizing path in one of $M_A$ or $M_B$, also having label $w$. This shows that if $w \in L(M_{A\cup B})$, then either $w \in L(M_A) = A$ or $w \in L(M_B) = B$ (of course, it might be in both, but this argument does not reveal this); in other words, if $w \in L(M_{A\cup B})$ then $w \in A \cup B$.

To show that $A \cup B \subseteq L(M_{A\cup B})$, it is enough to show that if $w \in A \cup B$ then $w \in L(M_{A\cup B})$ also. So let $w \in A$ and let $P_A$ be a recognizing path for $w$ in $M_A$. By preceding $P_A$ with the appropriate $\lambda$-edge from $\text{start}$ and following it with the $\lambda$-edge to $\text{final}$, we obtain a recognizing path in $M_{A\cup B}$, also having label $w$. This shows that if $w \in L(M_A) = A$ then $w \in L(M_{A\cup B})$ also. Similarly, if $w \in B$ then $w \in L(M_{A\cup B})$ too. Together, this gives that if $w \in A \cup B$ then $w \in L(M_{A\cup B})$. ■
If \( A \) and \( B \) are regular then so is \( A \circ B \). We show this using the NFA \( M_{AB} \) in Figure 7. \( M_{AB} \) is built using \( M_A \) and \( M_B \), NFAs with single final vertices recognizing \( A \) and \( B \), respectively. \( M_{AB} \) comprises a copy of \( M_A \) plus a copy \( M_B \), plus one additional edge. \( M_A \)'s start vertex is also \( M_{AB} \)'s start vertex, and \( M_B \)'s final vertex is \( M_{AB} \)'s only final vertex. Finally, the new edge, \( e \), joins \( M_A \)'s final vertex to \( M_B \)'s start vertex.

The idea of the construction is that a recognizing path in \( M_{AB} \) corresponds to recognizing paths in \( M_A \) and \( M_B \) joined by edge \( e \). It then follows that \( w \in L(M_{AB}) \) if and only if \( w \) is the concatenation of strings \( u \) and \( v \), \( w = uv \), with \( u \in A \) and \( v \in B \). We now show this more formally.

**Lemma 6** \( M_{AB} \) recognizes \( A \circ B \).

**Proof:** We show that \( L(M_{AB}) = A \circ B \) by showing that \( L(M_{AB}) \subseteq A \circ B \) and that \( A \circ B \subseteq L(M_{AB}) \).

To show that \( L(M_{AB}) \subseteq A \circ B \), it is enough to show that if \( w \in L(M_{AB}) \) then \( w \in A \circ B \) also. So let \( w \in L(M_{AB}) \) and let \( P \) be a recognizing path for \( w \) in \( M_{AB} \). Removing edge \( e \) from \( P \) creates two paths \( P_A \) and \( P_B \), with \( P_A \) being a recognizing path in \( M_A \) and \( P_B \) a recognizing path in \( M_B \). Let \( u \) and \( v \) be the path labels for \( P_A \) and \( P_B \), respectively. So \( u \in A \) and \( v \in B \). As \( e \) is a \( \lambda \)-edge, \( w = u\lambda v = uv \). This shows that \( w \in A \circ B \).

To show that \( A \circ B \subseteq L(M_{AB}) \), it is enough to show that if \( u \in A \) and \( v \in B \) then \( uv \in L(M_{AB}) \). So let \( u \in A \), \( v \in B \), let \( P_A \) be a recognizing path for \( w \) in \( M_A \), and let \( P_B \) be a recognizing path for \( v \) in \( M_B \). Then form a path \( P = P_A, e, P_B \) in \( M_{AB} \); clearly, this is a recognizing path. Further, it has label \( u\lambda v = uv \). So \( uv \in L(M_{AB}) \).

If \( A \) is regular then so is \( A^* \). We show this using the NFA \( M_{A^*} \) in Figure 8. \( M_{A^*} \) is built using \( M_A \), an NFA with a single final vertex recognizing \( A \). \( M_{A^*} \) comprises a copy of \( M_A \) plus a new start vertex, plus two additional \( \lambda \)-edges, \( e \) and \( f \). \( e \) joins
$M_A^*$’s start vertex to $M_A$’s start vertex, and $f$ joins $M_A$’s final vertex to $M_A^*$’s start vertex. $M_A^*$’s start vertex is also its final vertex.

The construction is based on the observation that removing all copies of $e$ and $f$ from a recognizing path in $M_A^*$ yields subpaths, $k$ of them say, each of which is a recognizing path in $M_A$. It then follows that a string $w$ is recognized by $M_A^*$ if and only if it is the concatenation of $k$ strings recognized by $M_A$, for some $k \geq 0$. Now we show this more formally.

**Lemma 7** $M_A^*$ recognizes $A^*$.

**Proof:** We show that $L(M_A^*) = A^*$ by showing that $L(M_A^*) \subseteq A^*$ and that $A^* \subseteq L(M_A^*)$.

To show that $L(M_A^*) \subseteq A^*$, it is enough to show that if $w \in L(M_A^*)$ then $w \in A^*$ also. So let $w \in L(M_A^*)$ and let $P$ be a recognizing path for $w$ in $M_A^*$. Removing all instances of edges $e$ and $f$ from $P$ creates $k$ subpaths, for some $k \geq 0$, where each subpath is a recognizing path in $M_A$. Let the path labels on these $k$ subpaths be $u_1, \ldots, u_k$, respectively. So $u_i \in A$, for $1 \leq i \leq k$. As $e$ and $f$ are $\lambda$-edges, $w = u_1 \lambda u_2 \lambda \cdots \lambda u_k = u_1 u_2 \cdots u_k$. Thus $w \in A^*$.

To show that $A^* \subseteq L(M_A^*)$, it is enough to show that if $w \in A^*$ then $w \in L(M_A^*)$. If $w \in A^*$ then $w = u_1 u_2 \cdots u_k$, with $u_1, u_2, \ldots, u_k \in A$, for some $k \geq 0$. As $u_i \in A$, there is an accepting path $P_i$ in $M_A$ for $u_i$. Let $P$ be the following path in $M_A^*$: $e, P_1, f, e, P_2, f, \cdots, e, P_k, f$ (for $k = 0$ we intend the path of zero edges). $P$ is a recognizing path in $M_A^*$, and it has label $u_1 u_2 \cdots u_k = w$, as $e$ and $f$ are $\lambda$-edges. Thus $w \in L(M_A^*)$.

**1.2 Every regular language is recognized by a DFA**

Let $N$ be an NFA. We show how to construct a DFA $M$ with $L(M) = L(N)$.
Recall that to implement an NFA we keep track of the set of currently reachable vertices. This is what \( M \) will do with its vertices or states, which we call superstates henceforth. Each of \( M \)'s superstates is a subset of the set \( Q \) of vertices or states in \( N \). So \( M \)'s collection (set) of superstates is the power set of \( Q \), \( 2^Q \).

The relation between superstates of \( M \) and sets of vertices in \( N \) is given by the following assertion.

**Assertion 8** On input \( w \), \( M \) reaches superstate \( R \) if and only if \( N \) can reach the set of states \( R \).

To achieve Assertion 8, we define the transition function \( \delta \) for \( M \) to be identical to the transition function \( \delta \) for \( N \). Note that this specifies where \( M \)'s edges go.

To make sure the assertion is correct initially, that is for \( w = \lambda \), we set \( M \)'s start superstate to be the set of \( N \)'s vertices that \( N \) can reach on input \( \lambda \). Finally, we set the final superstates of \( M \) to be those superstates that include one or more of the final vertices of \( N \): so if \( F \) is the set of \( N \)'s final vertices, then \( R \) is a final superstate for \( M \) exactly if \( R \cap F \neq \Phi \).

**Lemma 9** \( L(M) = L(N) \).

**Proof:** We begin by arguing by induction on the length of the input string read so far that Assertion 8 is true. The base case is for the empty string: the claim is true by construction. For the inductive step, if the claim is true for strings of length \( k \), then we argue that it is also true for strings of length \( k + 1 \). By the inductive hypothesis, on an input \( w \) of length \( k \), \( R \) is the superstate reached by \( M \) if and only if \( R \) is the set of vertices reachable by \( N \). But then, on input \( wa \), \( M \) reaches superstate \( \delta(R, a) \), and \( N \) can reach the set of vertices \( \delta(R, a) \), so the claim is true for strings of length \( k + 1 \) also.

It remains to consider which strings the two machines recognize. \( M \) recognizes input \( w \) if and only if on input \( w \) it reaches superstate \( R \) where \( R \cap F \neq \Phi \); and \( N \) recognizes input \( w \) if and only if it can reach a set of vertices \( S \) where \( S \cap F \neq \Phi \). But we have shown that \( S = R \). So the two machines recognize the same collection of strings, that is \( L(M) = L(N) \).

**Implementation Remark** The advantage of an NFA is that it may have far fewer vertices (states) than a DFA recognizing the same language, and thus use much less memory to store them. On the other hand, when running the machines, the NFA may be less efficient, as the set of reachable vertices had to be computed as the input is read, whereas in the DFA one just needs to follow a single edge. Which choice is better depends on the particulars of the language and the implementation environment.