Non-Deterministic Finite Automata (NFA)

In a DFA, the current state and the input symbol read, determine a unique next state. In an NFA, one or more or none next states are possible and in addition, a state may be changed without reading an input symbol (referred to as "\( \varepsilon \)-move").

![Diagram of NFA example]

**Note**

- More than one moves:

- \( \varepsilon \)-moves:

- Move missing

In state \( q_1 \) and input \( 1 \), the NFA can stay in state \( q_1 \) or change state to \( q_2 \).
On given input, an NFA has several possible computation paths, forming a computation tree.

E.g., on input 01011

Thus on input 01011 the NFA computation path
- Either has no move at some step, tantamount to REJECT
- OR ends in states $q_1$ or $q_2$ and REJECTS
- OR ends in state $q_4$ and ACCEPTS (since $q_4$ is designated as an accept state.)
Def An NFA N is said to accept input \( x \in \Sigma^* \) iff there exists at least one computation path of N on input \( x \) that ends in an accept state.

Note The above NFA accepts 101, 11, 01011, ... It does not accept 100, ... (verify!)

It is not difficult to see that it accepts precisely those strings that have 11 or 101 as a consecutive substring.

We'll see that every NFA N has an equivalent DFA M, in the sense that N and M accept precisely the same set of inputs, i.e., that \( L(N) = L(M) \) where \( L(N) \), the language recognized by N is accepted

\[
L(N) = \{ x \in \Sigma^* | N \text{ accepts } x, \text{ i.e. } N \text{ has at least one computation path on } x \text{ that accepts} \}.
\]
Constructing NFA is often easier.

Ex. Let $\Sigma = \{0, 1\}$,

$L = \{ x \mid \text{the fourth symbol of } x \text{ from the end is } 1 \}$.

$L$ is regular and the DFA recognizing it must have at least 16 states. However an NFA is easily constructed.

Ex. Let $\Sigma = \{ a \}$.

$L = \{ x \mid \text{ } |x| \text{ is multiple of 2 or 3} \}.$
Ex. let $\Sigma = \{0, 1\}$. $s_1s_2...s_k \in \Sigma^*$ be a fixed string/pattern,

$L = \{ x \mid x \text{ has } s_1s_2...s_k \text{ as consecutive substring} \}$

Formal Definition of NFA

Def An NFA $N$ is a 5-tuple $N = (Q, \Sigma, \delta, q_0, F)$ where

- $Q$ is a finite set of states.
- $\Sigma$ is alphabet.
- $q_0$ is the start state.
- $F \subseteq Q$ is the subset of accept states.
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$ is the transition function.
Note
- $S(q, a)$ is the subset of states on current state $q$ and input symbol $a$. This allows one or many or none possible moves, none if $S(q, a) = \emptyset$.
- $S(q, \varepsilon)$ allows $\varepsilon$-moves.

Example:

$\delta$

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>1</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>${q_1}$</td>
<td>${q_1, q_2}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>${q_3}$</td>
<td>$\emptyset$</td>
<td>${q_3}$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$\emptyset$</td>
<td>${q_4}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>${q_4}$</td>
<td>${q_4}$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

$Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$,
$F = \{q_4\}$. 
Acceptance by an NFA (formal definition)

**Def** Let $N = (Q, \Sigma, \delta, q_1, F)$ be an NFA and $x \in \Sigma^*$ be an input. $N$ accepts $x$ iff

- $x$ can be written as $x = x_1x_2 \ldots x_n$ where $x_i \in \Sigma \cup \{\varepsilon\}$ for $1 \leq i \leq n$ and
- there exist states $r_1, r_2, \ldots, r_{n+1} \in Q$ such that
  1. $r_1 = q_1$
  2. $r_{i+1} \in \delta(r_i, x_i)$ for $i = 1, 2, \ldots, n$
  3. $r_{n+1} \in F$.

**Def** $L(N) = \{x \in \Sigma^* | N$ accepts $x\}$.

Equivalence of DFAs and NFAs.

**Observation** Every DFA is also an NFA.

**Justification** Evident.

Formally, $M = (Q, \Sigma, q_1, \delta, F)$, a DFA, is also an NFA where $\delta(q, \varepsilon)$ is thought of as a singleton set for
all \( q \in Q, a \in \Sigma \) and \( S(q, \varepsilon) = \emptyset \).

We now focus on proving that:

**Theorem**  Every NFA has an equivalent DFA.

**Proof**  The proof consists of two steps:
- First, remove \( \varepsilon \)-moves from the NFA.
- Then, construct an equivalent DFA by the "subset construction".

**Step 1**: Removing \( \varepsilon \)-moves

**Idea**  Suppose our NFA is

\[
\sim \rightarrow q_1 \xrightarrow{0, \varepsilon} q_2 \xrightarrow{1} q_3
\]

We remove the \( \varepsilon \)-move \( q_1 \xrightarrow{\varepsilon} q_2 \). Since in state \( q_1 \) and input 1, one could have first moved to \( q_2 \) "for free" and then to \( q_3 \), we should add the move:

\[
\sim \rightarrow q_1 \xrightarrow{1} q_3
\]
Moreover, any input that makes the NFA end in state \( q_1 \) is accepted since one could then move to the accept state \( q_2 \) "for free." Thus if we were to remove the \( \varepsilon \)-move, we better make \( q_1 \) an accept state to preserve this "functionality." Hence, an equivalent NFA (without \( \varepsilon \)-moves) is:

```
(0, 1) \rightarrow (q_1) \rightarrow (q_2) \rightarrow (q_3)
```

**Actual Construction**

Given an NFA, order its set of states \( Q \) arbitrarily. Go over the states \( q \in Q \) in that order, one by one, and for each state \( q \in Q \):

Remove All \( \varepsilon \)-moves Incoming into \( q \)
Remove All $\varepsilon$-moves into (9).

1. Remove $\varepsilon$-self-loop $q$ if any.

2. $\forall q'$ such that $q' \neq q$ and $q' \xrightarrow{\varepsilon} q$,

\[ \{ \forall q'' \neq a \in \Sigma \cup \{\varepsilon\} \text{ such that } q' \xrightarrow{\varepsilon} q \xrightarrow{a} q'' \} \]

add the move $q' \xrightarrow{a} q''$.

- If $q$ is an accept state, make $q'$ also an accept state.

- Delete the $\varepsilon$-move $q' \xrightarrow{\varepsilon} q$.

Exercise: Convince yourself that the above operation leads to an equivalent NFA.

- once all incoming $\varepsilon$-moves into $q$ are removed, no incoming $\varepsilon$-moves into $q$ are added subsequently. (WHY ?!)
Note: The construction "automatically" gets rid of \( \varepsilon \)-cycles if any. E.g.,

\[
\begin{align*}
\varepsilon & \quad \rightarrow q_1 \\
\varepsilon & \quad \rightarrow q_2 \\
\varepsilon & \quad \rightarrow q_3 \\
\end{align*}
\]

We'll now assume that the given NFA has no \( \varepsilon \)-moves. E.g.,

\[
\begin{align*}
Q^{0,1} & \quad 
\rightarrow q_1 \quad \rightarrow q_2 \\
& \quad \rightarrow q_3 \quad \rightarrow q_4 \\
\end{align*}
\]

We wish to construct an equivalent DFA. The idea is to "remember" the set of all states that the NFA could possibly
be in, after reading any input (prefix). E.g. on input 01010, the subset changes as:

\[
\begin{array}{c}
\{q_1\} \\
\downarrow 0 \\
\{q_1\} \\
\downarrow 1 \\
\{q_1, q_2\} \\
\downarrow 0 \\
\{q_1, q_3\} \\
\downarrow 1 \\
\{q_1, q_2, q_4\} \\
\downarrow 0 \\
\{q_1, q_3, q_4\}
\end{array}
\]

Thus, after reading 01010, the NFA could have been in any of the states \(\{q_1, q_3, q_4\}\). In particular, it could have been in the accept state \(q_4\), and so the NFA accepts the input 01010.
So we can simulate the NFA by a DFA such that:

- States of the DFA are all possible subsets of the set of states of the NFA.
- The DFA "remembers" the subset of all states that the NFA could be in.
- Transitions of DFA are defined as:

\[ R \xrightarrow{0} R', \quad R \xrightarrow{1} R'' \]

where \( R, R', R'' \subseteq Q \), \( Q \) is the set of states of NFA, and (say) \( R' \) is all states that the NFA can move to on input 0 from some state in \( R \).

- If \( R \) contains some accept state of NFA, then \( R \) is designated as accept state of DFA.
Formal construction

Let \( N = (Q, \Sigma, \delta, q_1, F) \) be an NFA with no \( \epsilon \)-moves. A DFA \( M = (Q', \Sigma, \delta', q'_1, F') \) that is equivalent to \( N \) is constructed as follows:

- \( Q' = \mathcal{P}(Q) \).
- \( q'_1 = \{ q_1 \} \).
- \( F' = \{ R \mid R \subseteq Q, R \cap F \neq \emptyset \} \).
- For \( R \subseteq Q \) and \( a \in \Sigma \),
  \[
  s'(R, a) = \bigcup_{q \in R} s(q, a).
  \]
We can first remove the \( \varepsilon \)-move and get

Then the subset construction gives a DFA:

**Exercise** complete all the transitions of this DFA.

**Note** Given \( k \)-state NFA, the construction gives \( 2^k \)-state DFA.