An NFA $M$ is a tuple $(Q, \Sigma, \Delta, s, F)$, where $Q$, $\Sigma$, $s$, $F$ are the same as in the definition of DFA, and

$$\Delta: Q \times (\Sigma \cup \{e\}) \rightarrow 2^Q$$

The concept of computation path is the same as for DFA, but the concept of acceptance is different, which is a generalization of the acceptance notion of the DFA: A string $x$ is said to be acceptable by an NFA $M$ if there exists one computation path of $M$ on $x$ that ends at a final state.

**Advantages of using an NFA:**

- It’s easier to build an NFA to accept a given regular language.

Definition: A regular language is a language accepted by a DFA.

**Q:** Is NFA more powerful than DFA. I.e. is there a language $L$ s.t. $L$ is accepted by an NFA, but not accepted by any DFA?

**A:** NO.

We will device an algorithm that takes a regular expression (representing a regular language) as input and generates a DFA, with a minimum number of states, which accepts the regular language as output.

**We are going to solve this problem in the following steps:**

1. Convert an NFA to an equivalent DFA
2. Build an NFA on a given regular expression
3. Given a DFA, obtain an equivalent DFA with a minimum number of states

**NFA:**

![NFA Diagram](Fig.1)
E-closure of a given state,
$$E[p] = \{q \mid q \text{ can be reached from } p \text{ via a sequence of e-links}\} \cup \{p\}$$

$$E[A] = \{A, B, E\}$$
$$E[B] = \{B, E\}$$
$$E[C] = \{C\}$$
$$E[D] = \{D, E\}$$
$$E[E] = \{E\}$$

E-closure of a given set S:
$$E[S] = \{q \mid q \text{ can be reached via a sequence of e-moves from a state in } S\} \cup S$$
E.g. $$E[B, D] = \{B, D, E\}$$

We are going to look at E-closure and “collect” (“compress”) a set of states into one state.

We start from the E-closure of the initial state of the NFA $$E[A] = \{A, B, E\}$$; we make it as a new initial state.

Any such “big” state is a final state if it contains a final state of the given NFA in it.

<table>
<thead>
<tr>
<th>δ</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>*[C, E]</td>
<td></td>
<td>E[E]= [E]</td>
</tr>
<tr>
<td>*[D, E]</td>
<td></td>
<td>E[E]= [E]</td>
</tr>
<tr>
<td>*[E]</td>
<td></td>
<td>E[E]= [E]</td>
</tr>
</tbody>
</table>

**Fig.2 Table**

**DFA:**

```
[ A, B, E ]
  ↓ a
[ C, E ]   a, b
  ↓
[ E ]
  ↓ a
[ D, E ]
```

**Fig3**
Algorithm that converts a given NFA, $N$, to an equivalent DFA, $M$:

1. Let the E-closure of the initial state of $N$ be the initial state of $M$.
2. For each new state of $M$, $[p_1, p_2, \ldots, p_k]$, find the set of states that can be reached by one more under a given symbol from each of the states $p_i$:
   $$\{q \mid p_i \rightarrow q, p_i \in \{ p_1, \ldots, p_k \}\}$$
   Then we calculate the E-closure of this set, say it’s $[q_1, q_2, \ldots, q_l]$.
3. Repeat step 2 until no more new states are generated.
4. Make every state in $M$ with a state in $F$ of $N$ as a final state of $M$. This algorithm stops because $2^Q$ is finite.

Example: Fig2.20 pp. 28

NFA:

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rightarrow[s]$</td>
<td>$E[q]=[q, r, t]$</td>
<td>$E[r]=[r, t]$</td>
</tr>
<tr>
<td>$[q, r, t]$</td>
<td>$E[t]=[t]$</td>
<td>$E[s, t]=[s, t]$</td>
</tr>
<tr>
<td>$[r, t]$</td>
<td>$E[t]=[t]$</td>
<td>$E[t]=[t]$</td>
</tr>
<tr>
<td>$[t]$</td>
<td>$\phi$</td>
<td>$E[t]=[t]$</td>
</tr>
<tr>
<td>$[s, t]$</td>
<td>$E[q]=[q, r, t]$</td>
<td>$E[r]=[r, t]$</td>
</tr>
<tr>
<td>$\phi$</td>
<td>$\phi$</td>
<td>$\phi$</td>
</tr>
</tbody>
</table>
DFA: