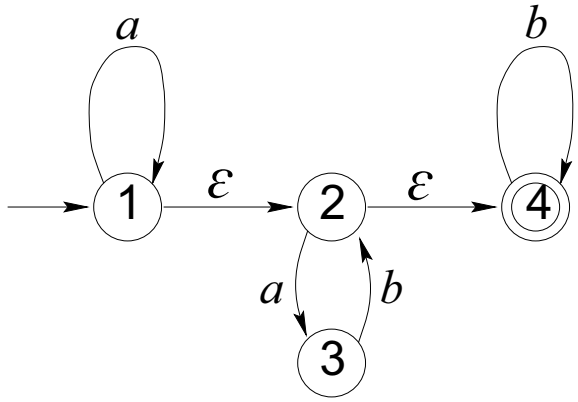
NFA with ε-moves

In the next section, it turns out to be useful to allow NFA with **spontaneous transitions**. When an NFA executes a spontaneous transition (called an **ε-move**) it changes its state without reading any input letter. Any number of consecutive ε-moves are allowed.

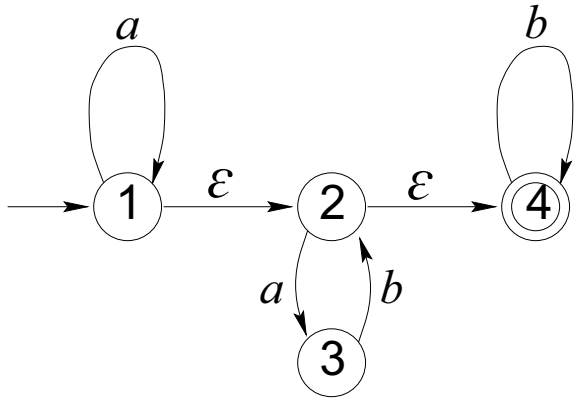
Here's an example of an NFA with ε-moves:



NFA with ε-moves

In the next section, it turns out to be useful to allow NFA with **spontaneous transitions**. When an NFA executes a spontaneous transition (called an **ε-move**) it changes its state without reading any input letter. Any number of consecutive ε-moves are allowed.

Here's an example of an NFA with ε-moves:



The automaton accepts any sequence of a's followed by any repetition of ab's followed by any number of b's:

$$L(A) = \{a^i(ab)^j b^k \mid i, j, k \geq 0\}.$$

Formally, an NFA with ε-moves (or **ε-NFA** for short) is $A = (Q, \Sigma, \delta, q_0, F)$ where Q, Σ, q_0 and F are as before, and δ is a function

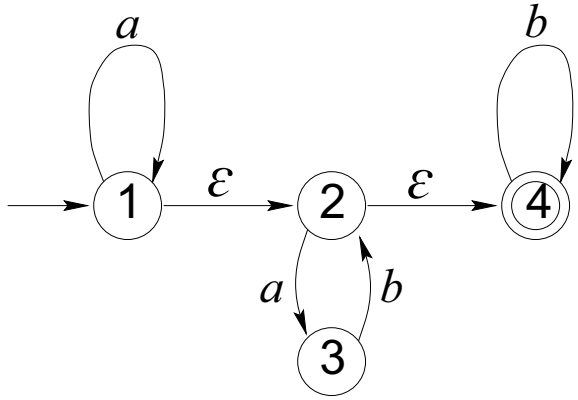
$$\mathbf{Q} \times (\mathbf{\Sigma} \cup \{\varepsilon\}) \longrightarrow \mathbf{2^Q}$$

that specifies for each state q transitions with all input letters a, and the empty word ε.

- $\delta(q, a)$ is the set of all states p such that there is a transition from q to p with input a, and

- $\delta(q, \varepsilon)$ is the set of states p such that there is a spontaneous transition from q to p.

Example.

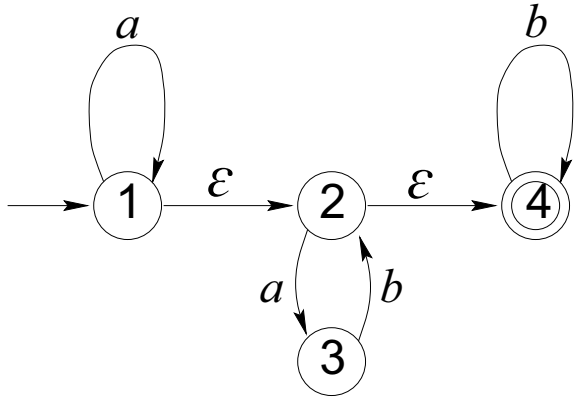


The transition function

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \longrightarrow 2^Q$$

is

Example.



The transition function

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \longrightarrow 2^Q$$

is

	a	b	ε
1	$\{1\}$	$\emptyset$	$\{2\}$
2	$\{3\}$	$\emptyset$	$\{4\}$
3	$\emptyset$	$\{2\}$	$\emptyset$
4	$\emptyset$	$\{4\}$	$\emptyset$

We **extend** δ to function $\hat{\delta}$ that gives reached states for all input words:

$$\hat{\delta} : Q \times \Sigma^* \longrightarrow 2^Q$$

Set $\hat{\delta}(q, w)$ contains all possible states after the automaton reads input word w, starting at state q. When processing w any number of ε-moves may be used.

We **extend** δ to function $\hat{\delta}$ that gives reached states for all input words:

$$\hat{\delta} : Q \times \Sigma^* \longrightarrow 2^Q$$

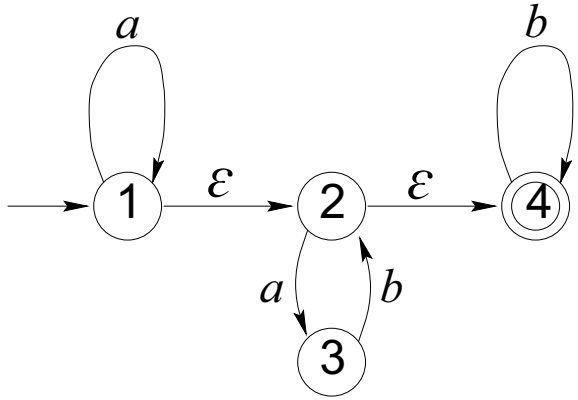
Set $\hat{\delta}(q, w)$ contains all possible states after the automaton reads input word w, starting at state q. When processing w any number of ε-moves may be used.

In the recursive definition of $\hat{\delta}$ we use the concept of an **ε-closure**. For any state q we let

$$\varepsilon\text{-CLOSURE } (q)$$

be the set of states reached from q using only ε-moves.

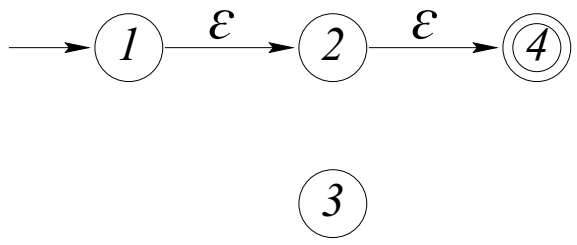
Example.



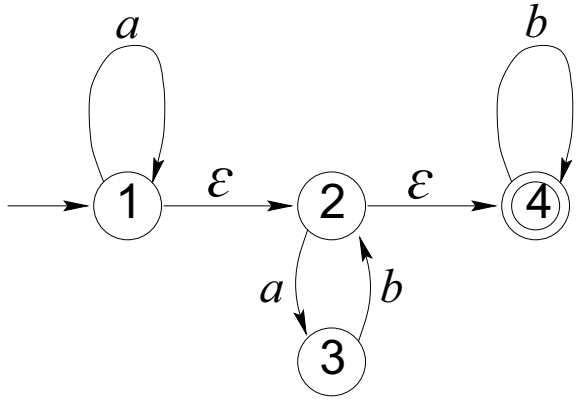
The ε-closures of the states are

ε-CLOSURE (1) =
ε-CLOSURE (2) =
ε-CLOSURE (3) =
ε-CLOSURE (4) =

The set ε-CLOSURE(q) contains exactly the reachable states from q in the graph



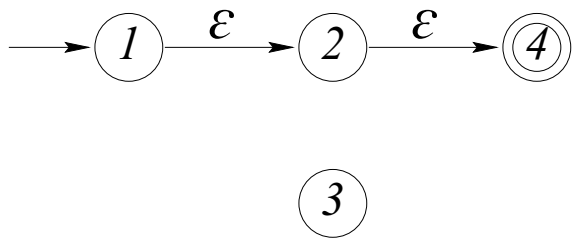
Example.



The ε-closures of the states are

$$\begin{aligned}
\varepsilon\text{-CLOSURE }(1) &= \{1, 2, 4\} \\
\varepsilon\text{-CLOSURE }(2) &= \{2, 4\} \\
\varepsilon\text{-CLOSURE }(3) &= \{3\} \\
\varepsilon\text{-CLOSURE }(4) &= \{4\}
\end{aligned}$$

The set ε-CLOSURE(q) contains exactly the reachable states from q in the graph



For any **subset of states** $S \subseteq Q$ we denote ε-CLOSURE (S) for the set of states reachable from any element of S using only ε-moves:

$$\varepsilon\text{-CLOSURE } (S) = \bigcup_{q \in S} \varepsilon\text{-CLOSURE } (q).$$

Here is a recursive definition of $\hat{\delta}(q, w)$:

1. For every state q

$$\hat{\delta}(q, \varepsilon) = \varepsilon\text{-CLOSURE } (q).$$

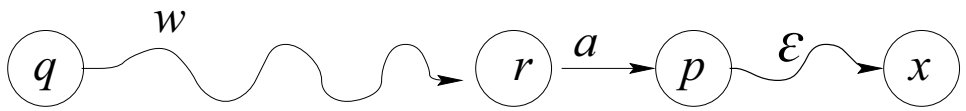
(By definition, the ε-CLOSURE consists of all states reachable with ε-moves.)

2. For every state q, word w and letter a

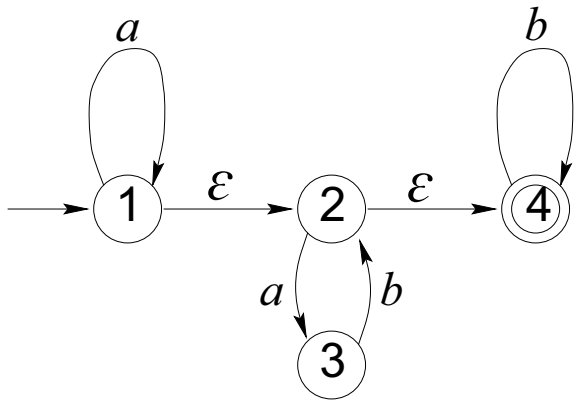
$$\hat{\delta}(q, wa) = \varepsilon\text{-CLOSURE } \left(\bigcup_{r \in \hat{\delta}(q,w)} \delta(r, a) \right) = \varepsilon\text{-CLOSURE } (S)$$

where

$$S = \{p \mid \exists\ r \in \hat{\delta}(q, w) \ : \ p \in \delta(r, a)\}.$$

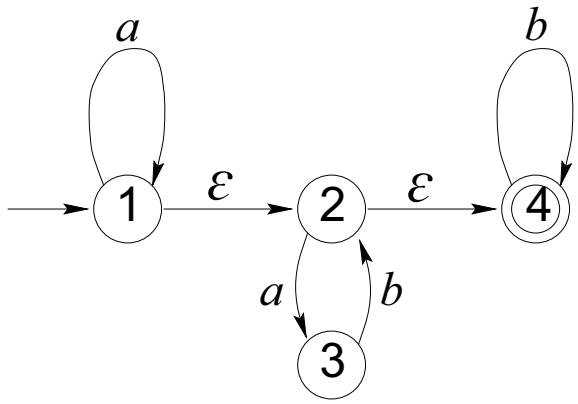


Example.



$$
\begin{aligned}
\hat{\delta}(1,\varepsilon) &= \{1,2,4\} \\
\hat{\delta}(2,\varepsilon) &= \{2,4\} \\
\hat{\delta}(3,\varepsilon) &= \{3\} \\
\hat{\delta}(4,\varepsilon) &= \{4\} \\
\hat{\delta}(1,a) &= \\
\hat{\delta}(1,ab) &= \\
\hat{\delta}(1,aba) &= \\
&\ldots
\end{aligned}
$$

Example.



$$
\begin{aligned}
\hat{\delta}(1,\varepsilon) &= \{1,2,4\} \\
\hat{\delta}(2,\varepsilon) &= \{2,4\} \\
\hat{\delta}(3,\varepsilon) &= \{3\} \\
\hat{\delta}(4,\varepsilon) &= \{4\} \\
\hat{\delta}(1,a) &= \{1,2,3,4\} \\
\hat{\delta}(1,ab) &= \{2,4\} \\
\hat{\delta}(1,aba) &= \{3\} \\
&\ldots
\end{aligned}
$$

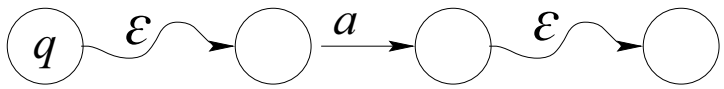
The language recognized by ε-NFA A is

$$L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

The values of $\hat{\delta}(q, a)$ for single letters a are of special interest to us. First of all, they are **not** necessarily identical to $\delta(q, a)$, so we cannot remove the hat as we did with DFA and NFA:

- $\delta(q, a)$ contains only states you can reach with one transition labeled by a. It does not allow using ε-moves.

- $\hat{\delta}(q, a)$ contains all states you can reach from q by doing any number of ε-moves and one a-transition, in any order:

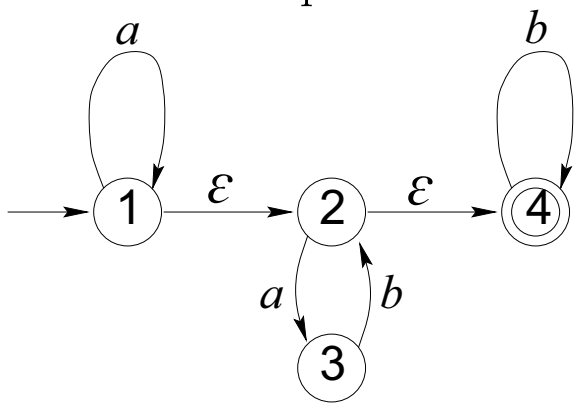


Introducing spontaneous transitions does not increase the power of NFA. Next we show that ε-NFA accept exactly the same regular languages as DFA and NFA:

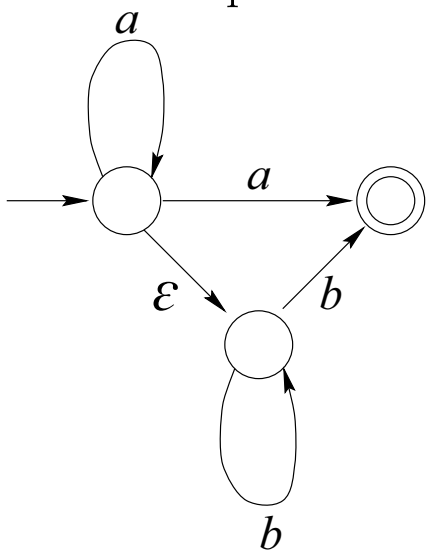
Theorem. For any given ε-NFA A one can effectively construct an NFA A' such that $L(A') = L(A)$.

Proof.

Example. Let us construct an NFA equivalent to



Example. Let us construct an NFA equivalent to



Now we know how to convert any ε-NFA into an equivalent NFA, and how to convert any NFA into an equivalent DFA, so we can convert any ε-NFA into a DFA. All three automata models recognize the same languages.

Regular expressions

Regular expressions provide a different technique to define languages. Instead of being accepting devices such as finite automata they are **generating** devices.

By **Kleene's theorem** (proved later) regular expressions define exactly regular languages (=the same languages recognized by finite automata).

We need some **operations** on languages.

• The **concatenation** L_1L_2 of languages L_1 and L_2 is the language containing all words obtained by concatenating a word from L_1 and a word from L_2:

$$L_1L_2 = \{uv \mid u \in L_1 \text{ and } v \in L_2\}$$

For example

$\{ab, b\}\{aa, ba\} =$

We need some **operations** on languages.

• The **concatenation** L_1L_2 of languages L_1 and L_2 is the language containing all words obtained by concatenating a word from L_1 and a word from L_2:

$$L_1L_2 = \{uv \mid u \in L_1 \text{ and } v \in L_2\}$$

For example

$$\{ab, b\}\{aa, ba\} = \{abaa, abba, baa, bba\}$$

• For every $n \geq 0$ we define L^n to be the set of words obtained by concatenating n words from language L. A precise recursive definition:

$$\begin{aligned} L^0 &= \{\varepsilon\} \\ L^n &= L^{n-1}L \text{ for every } n \geq 1. \end{aligned}$$

For example,

$\{ab, b\}^3 =$

• For every $n \geq 0$ we define L^n to be the set of words obtained by concatenating n words from language L. A precise recursive definition:

$$\begin{aligned} L^0 &= \{\varepsilon\} \\ L^n &= L^{n-1}L \text{ for every } n \geq 1. \end{aligned}$$

For example,

$\{ab, b\}^3 = \{ababab, ababb, abbab, abbb, babab, babb, bbab, bbb\}$

• The **Kleene closure** L^* of language L is the set containing words obtained by concatenating any number of words from L together:

$$L^* = \bigcup_{i=0}^{\infty} L^i.$$

For example,

$$\{ab, b\}^* = \{\varepsilon, ab, b, abab, abb, bab, bb, ababab, ababb, \dots\}$$

Note that if $L = \Sigma$ then $L^* = \Sigma^*$ is the set of all words over alphabet Σ. We have already used this notation earlier!

- The **positive closure** L^+ of L is

$$L^+ = \bigcup_{i=1}^{\infty} L^i,$$

i.e., concatenations of one or more words from L. For example,

$$\{ab, b\}^+ = \{ab, b, abab, abb, bab, bb, ababab, ababb, \dots\}.$$

We have always

$$L^* = L^+ \cup \{\varepsilon\}.$$

Also, we have always

$$L^+ = LL^*.$$

- When are languages L^* and L^+ identical ?
- When are languages L^* and L^+ finite ?
- What language is $\emptyset^*$? What about $\emptyset^+$?

- The **union** $L_1 \cup L_2$ of languages L_1 and L_2 is just the usual union as sets.

Next we define **regular expressions** over alphabet Σ. They are syntactic expressions that represent certain languages. If r is a regular expression then we denote the language it represent as $L(r)$.

Next we define **regular expressions** over alphabet Σ. They are syntactic expressions that represent certain languages. If r is a regular expression then we denote the language it represent as $L(r)$.

$\emptyset$ is a regular expression representing the empty language.

Next we define **regular expressions** over alphabet Σ. They are syntactic expressions that represent certain languages. If r is a regular expression then we denote the language it represent as $L(r)$.

$\emptyset$ is a regular expression representing the empty language.

ε is a regular expression and it represents the singleton language $\{\varepsilon\}$.

Next we define **regular expressions** over alphabet Σ. They are syntactic expressions that represent certain languages. If r is a regular expression then we denote the language it represent as $L(r)$.

$\emptyset$ is a regular expression representing the empty language.

ε is a regular expression and it represents the singleton language $\{\varepsilon\}$.

Every letter a of Σ is a regular expression representing the singleton language $\{a\}$.

Next we define **regular expressions** over alphabet Σ. They are syntactic expressions that represent certain languages. If r is a regular expression then we denote the language it represent as $L(r)$.

$\emptyset$ is a regular expression representing the empty language.

ε is a regular expression and it represents the singleton language $\{\varepsilon\}$.

Every letter a of Σ is a regular expression representing the singleton language $\{a\}$.

If r and s are arbitrary regular expressions then $(r+s)$, (rs) and (r^*) are regular expressions. If $L(r) = R$ and $L(s) = S$ then

$$\begin{aligned} L(r+s) &= R \cup S \\ L(rs) &= RS \\ L(r^*) &= R^* \end{aligned}$$

We may remove parentheses from regular expressions using the following precedence rules:

- the Kleene star * has highest precedence,
- the concatenation has second highest precedence,
- the union + has lowest precedence.

We may remove parentheses from regular expressions using the following precedence rules:

- the Kleene star * has highest precedence,
- the concatenation has second highest precedence,
- the union + has lowest precedence.

Because concatenation and union are associative, we may

- simplify $r(st)$ and $(rs)t$ into rst, and
- simplify $r + (s + t)$ and $(r + s) + t$ into $r + s + t$.

Example.

$$(((ab)^*) + (a(ba)))((a^*)b) = ((ab)^* + aba)a^*b.$$

Often we do not distinguish between a regular expression r and its language $L(r)$. We simplify notations by talking about language r.

Other shorthand notations:

- Expression rr^* may be denoted as r^+.
- Expression $\overbrace{rr\ldots r}^{n}$ may be abbreviated as r^n.

Example. Construct a regular expression for the following languages over the alphabet $\Sigma = \{a, b\}$:

1. $L = \{ab, ba\}$.

2. All words of Σ.

3. All words that start with a and end in b.

4. All words that contain aba as a subword.

5. Words that start with ab and end in ba.

6. Words in which some consecutive b's are separated by an even number of a's.

Example. Conversely, describe in English the following languages:

1. $a^*(ab)^*b^*$

2. $(ab + b)^*$

3. $(\varepsilon + b)(ab)^*(\varepsilon + a)$