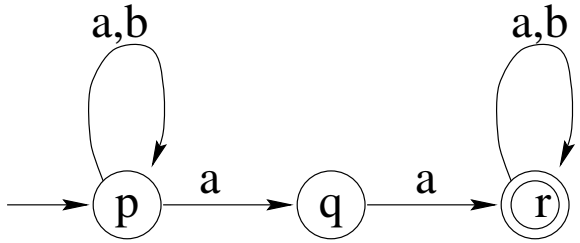
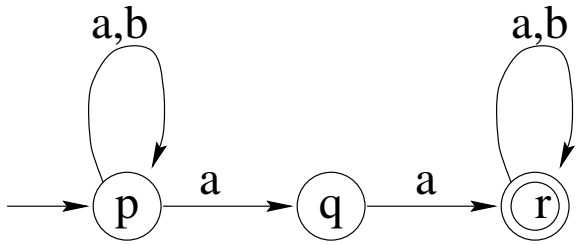
Nondeterministic finite automata (NFA)

Nondeterministic finite automata are generalizations of DFA. Instead of exactly one outgoing transition from each state by every input letter, NFA allow **several outgoing transitions** at the same time. A word is accepted by the NFA if **some** choice of transitions takes the machine to a final state. Some other choices may lead to a non-final state, but the word is accepted as long as there exists at least one accepting computation path in the automaton.

An example of a transition diagram of an NFA:

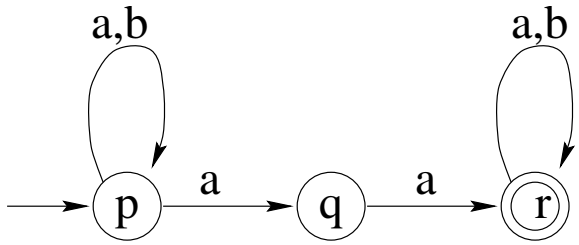


There are two transitions from state p with input letter a, into states p and q. Note also that there are no transitions from state q with input letter b. The number of transitions may be zero, one or more.



NFA may have several different computations for the same input word. Let us take for example word

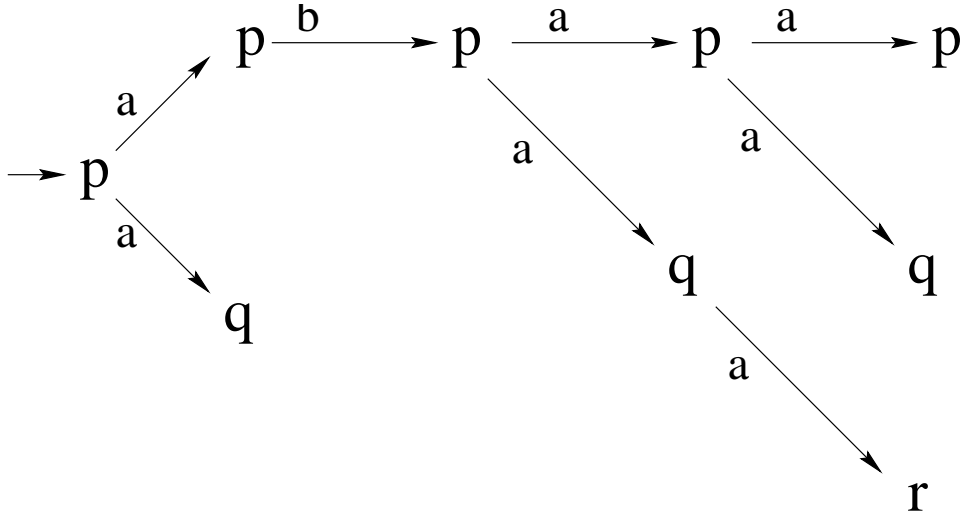
$$w = abaa.$$



NFA may have several different computations for the same input word. Let us take for example word

$$w = abaa.$$

The following computation tree summarizes all possible computations with input $abaa$:



Precise definition: An NFA $A = (Q, \Sigma, \delta, q_0, F)$ is specified by 5 items:

- State set Q,
- input alphabet Σ,
- initial state q_0, and
- final state set F

all have the same meaning as for a DFA.

- The transition function δ is defined differently. It gives for each state q and input letter a a **set** $\delta(q, a)$ of possible next states.

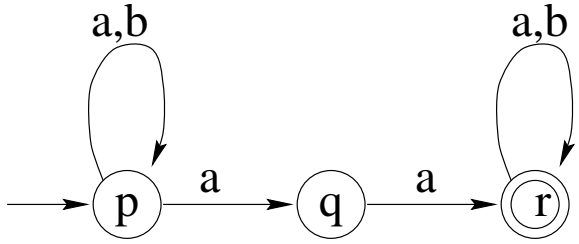
Using the power set notation

$$2^Q = \{S \mid S \subseteq Q\}$$

we can write

$$\delta : Q \times \Sigma \longrightarrow 2^Q.$$

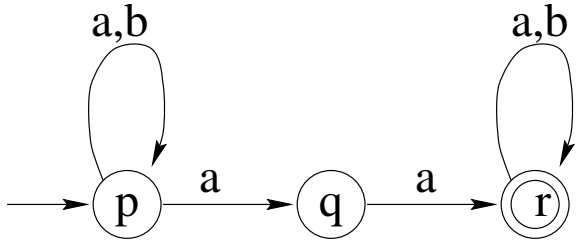
Example. The transition function δ of



is given by the table

	a	b
p	$\{p, q\}$	$\{p\}$
q	$\{r\}$	$\emptyset$
r	$\{r\}$	$\{r\}$

Example. The transition function δ of



is given by the table

	a	b
p	$\{p, q\}$	$\{p\}$
q	$\{r\}$	$\emptyset$
r	$\{r\}$	$\{r\}$

What is the language recognized by the sample NFA ? The language consists of all words for which there exists at least one accepting computation path.

Let us **extend** the meaning of the transition function δ the same way we did for DFA. We define

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

such that $\hat{\delta}(q, w)$ is the set of all states the machine can reach from state q reading input word w.

Let us **extend** the meaning of the transition function δ the same way we did for DFA. We define

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

such that $\hat{\delta}(q, w)$ is the set of all states the machine can reach from state q reading input word w.

The exact recursive definition goes like this:

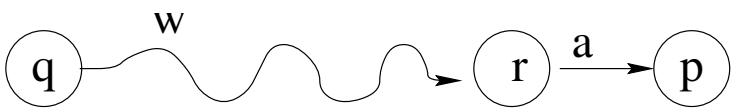
1. For every state q

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

(No state is changed if no input is read.)

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

$$\hat{\delta}(q, wa) = \bigcup_{r \in \hat{\delta}(q,w)} \delta(r, a) = \{p \mid \text{there exists state } r \text{ such that } r \in \hat{\delta}(q, w) \text{ and } p \in \delta(r, a) \}.$$

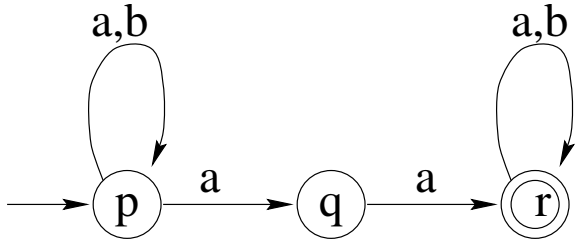


On single symbols δ and $\hat{\delta}$ have identical values:

$$\delta(q, a) = \hat{\delta}(q, a).$$

Therefore there is no risk of confusion if we drop the hat and write simply δ instead of $\hat{\delta}$.

Example. In our sample NFA



$$\begin{aligned}
\delta(p, a) &= \{p, q\}, \\
\delta(p, ab) &= \{p\}, \\
\delta(p, aba) &= \{p, q\}, \\
\delta(p, abaa) &= \{p, q, r\}.
\end{aligned}$$

The language recognized by NFA $A = (Q, \Sigma, \delta, q_0, F)$ is

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

(Words w such that there is a final state among the states $\delta(q_0, w)$ reachable from the initial state q_0 on input w.)

Let us design NFA over alphabet $\Sigma = \{a, b\}$ that recognize the following languages:

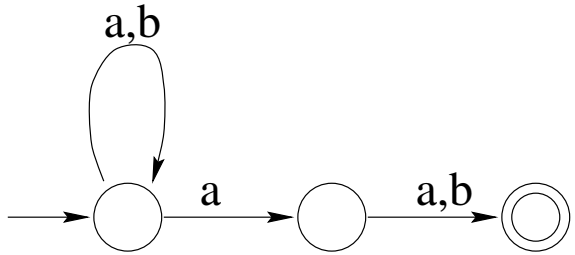
1. Words that end in ab.

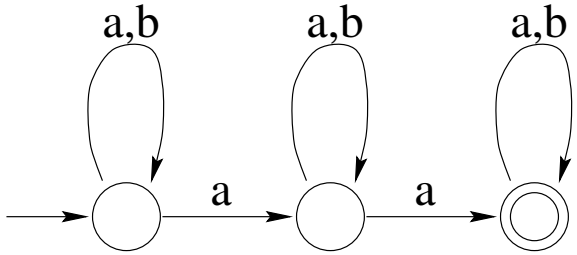
2. Words that contain aba as a subword.

3. Words that start with ab and end in ba

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

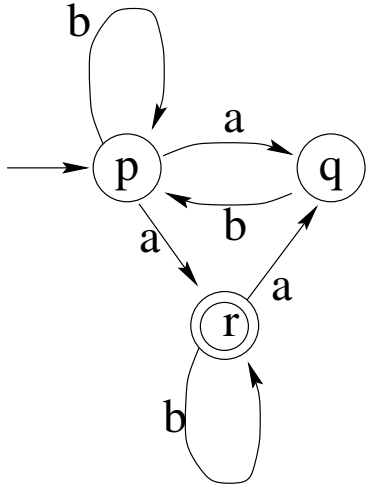
And the inverse problem: Describe in simple English sentence the words accepted by the following NFA:





Question: How does one go about checking if a given NFA accepts a given input word w ?

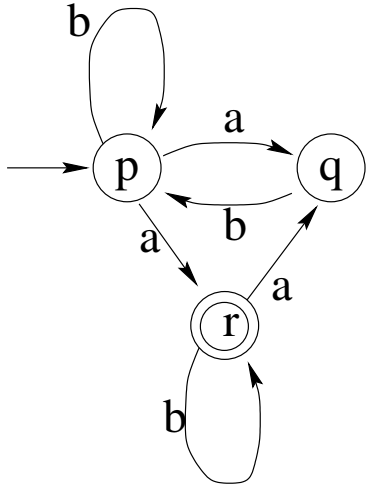
For example, how would one check if the NFA



accepts input $w = abbaabb$?

Question: How does one go about checking if a given NFA accepts a given input word w ?

For example, how would one check if the NFA

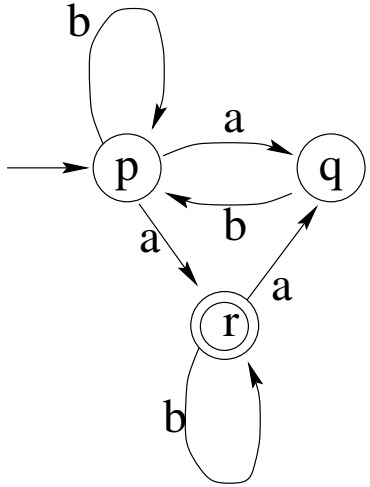


accepts input $w = abbaabb$?

One alternative is to try all possible computation paths for w and see if any of them ends in an accepting state. But the number of paths may be very large, and grow exponentially with the length of the input!

Question: How does one go about checking if a given NFA accepts a given input word w ?

For example, how would one check if the NFA

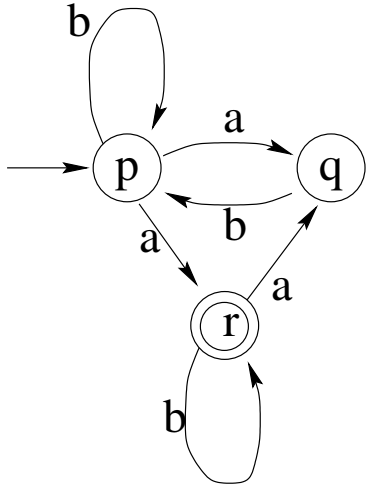


accepts input $w = abbaabb$?

A **better way** is to scan the input only once, and keep track of the **set** of possible states. For example, with our NFA and input $w = abbaabb$ we have:

Question: How does one go about checking if a given NFA accepts a given input word w ?

For example, how would one check if the NFA



accepts input $w = abbaabb$?

A **better way** is to scan the input only once, and keep track of the **set** of possible states. For example, with our NFA and input $w = abbaabb$ we have:

$$\{p\} \xrightarrow{a} \{q,r\} \xrightarrow{b} \{p,r\} \xrightarrow{b} \{p,r\} \xrightarrow{a} \{q,r\} \xrightarrow{a} \{q\} \xrightarrow{b} \{p\} \xrightarrow{b} \{p\}$$

The word is not accepted because one can only reach p which is not a final state.

Intuitively it may seem that NFA can recognize some languages that no DFA can recognize. But this is not the case: NFA and DFA recognize exactly the **same family of languages** (the regular languages).

Intuitively it may seem that NFA can recognize some languages that no DFA can recognize. But this is not the case: NFA and DFA recognize exactly the **same family of languages** (the regular languages).

1) First, any language recognized by some DFA is also recognized by some NFA; namely the same automaton. (Every DFA is also an NFA.)

Intuitively it may seem that NFA can recognize some languages that no DFA can recognize. But this is not the case: NFA and DFA recognize exactly the **same family of languages** (the regular languages).

1) First, any language recognized by some DFA is also recognized by some NFA; namely the same automaton. (Every DFA is also an NFA.)

2) Conversely, every language that is recognized by an NFA is also recognized by some DFA. To prove this we show how to construct a DFA that is equivalent to a given NFA, *i.e.*, it recognizes the same language.

Intuitively it may seem that NFA can recognize some languages that no DFA can recognize. But this is not the case: NFA and DFA recognize exactly the **same family of languages** (the regular languages).

1) First, any language recognized by some DFA is also recognized by some NFA; namely the same automaton. (Every DFA is also an NFA.)

2) Conversely, every language that is recognized by an NFA is also recognized by some DFA. To prove this we show how to construct a DFA that is equivalent to a given NFA, *i.e.*, it recognizes the same language.

The idea of the proof is to keep track of all possible states that the NFA can be in after reading the input. So we construct a DFA whose states are **sets of states of the original NFA**.

The states of the new DFA will be sets

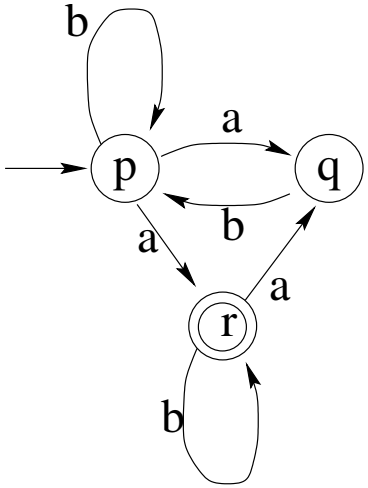
$$\{q_1, q_2, \ldots, q_k\}$$

where $q_1, q_2, \ldots, q_k$ are states of the NFA.

The interpretation of the state $\{q_1, q_2, \ldots, q_k\}$: Input w takes the DFA into state $\{q_1, q_2, \ldots, q_k\}$ if and only if $q_1, q_2, \ldots, q_k$ are precisely the states one can reach with input w in the NFA.

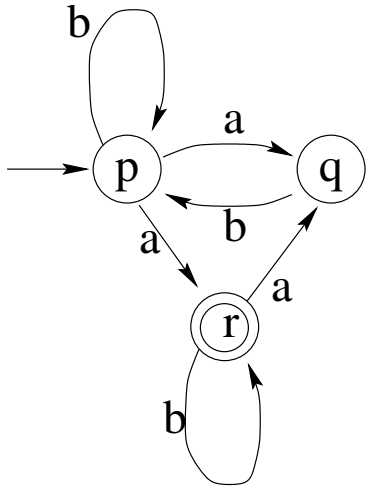
Before the general case, let us look at one example

Example. Let us construct a DFA that recognizes the same language as our sample NFA

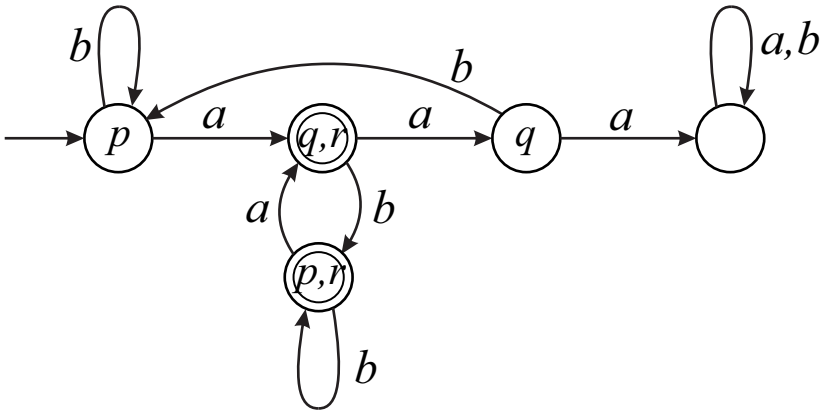


Before the general case, let us look at one example

Example. Let us construct a DFA that recognizes the same language as our sample NFA



The construction is called the **powerset** construction.



Remark. This is just one DFA that recognizes the same language; it is not necessarily the smallest one.

Let us prove the general case:

Theorem (The powerset construction). Given an NFA A one can effectively construct a DFA A' such that $L(A) = L(A')$.

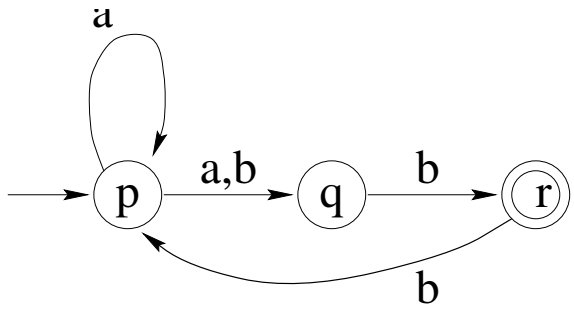
Proof.

Let us prove the general case:

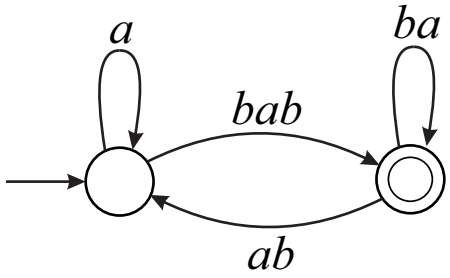
Theorem (The powerset construction). Given an NFA A one can **effectively** construct a DFA A' such that $L(A) = L(A')$.

Remark. "Effective construction" means that there is an algorithm (=mechanical procedure) to construct the DFA A' when the NFA A is given as input. The construction mechanically processes the NFA A, without any need to understand what A does. This is a stronger statement than simply stating that "there exists" a DFA that is equivalent to A.

Let us practice the powerset construction with the following NFA:



A simple extension of NFA: Allow transitions with words instead of just letters.

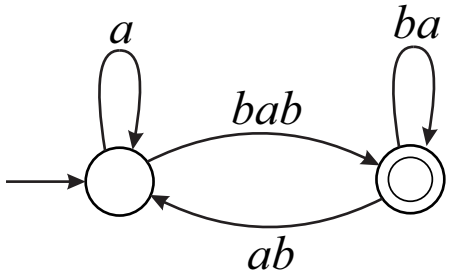


For example

$$babbaabbab = bab \cdot ba \cdot ab \cdot bab$$

is accepted.

A simple extension of NFA: Allow transitions with words instead of just letters.

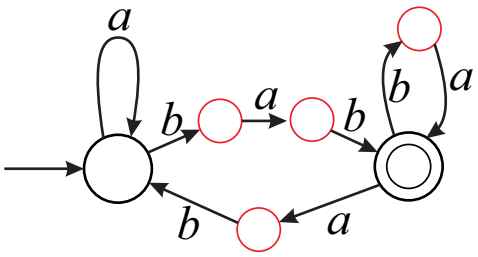


For example

$$babbaabbab = bab \cdot ba \cdot ab \cdot bab$$

is accepted.

Transitions with longer words can cut into single-letter transitions by adding new states:



The construction cannot be used for transitions with the empty word ε.