

Regular expressions can define exactly the same languages as DFA, NFA and ε -NFA, *i.e.*, the regular languages.

Theorem (Kleene 1956). Language L is recognized by some DFA (NFA, ε -NFA) if and only if there exists a regular expression for it.

Proof.

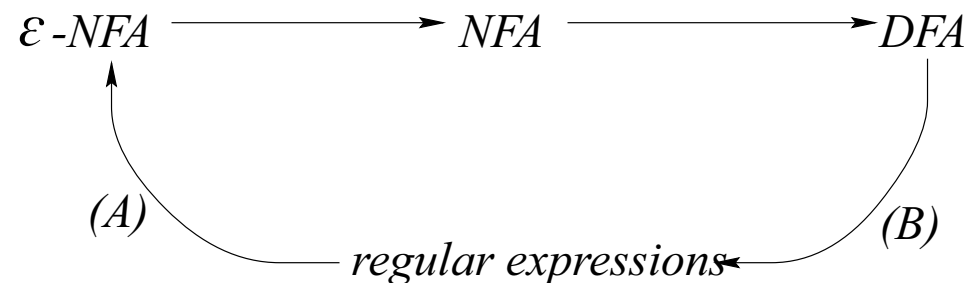
Theorem (Kleene 1956). Language L is recognized by some DFA (NFA, ε -NFA) if and only if there exists a regular expression for it.

Proof. To prove the theorem we show two directions:

(A) We show how to effectively construct for any given regular expression an equivalent ε -NFA.

(B) We show how to effectively construct for any given DFA an equivalent regular expression.

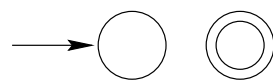
Together with our earlier proofs ε -NFA $\longrightarrow$ NFA and NFA $\longrightarrow$ DFA, constructions (A) and (B) allow us to transform any of the 4 devices into each other:



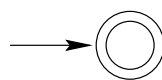
(A) Regular expression $\longrightarrow$ ε -NFA.

By induction on the size of the regular expression r we construct an ε -NFA A that has a **single final state** such that $L(A) = L(r)$.

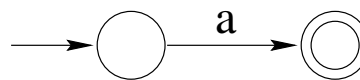
1. Base case. Below are diagrams for automata that recognize $\emptyset$, $\{\varepsilon\}$ and $\{a\}$. Every machine has exactly one final state:



$r = \emptyset$



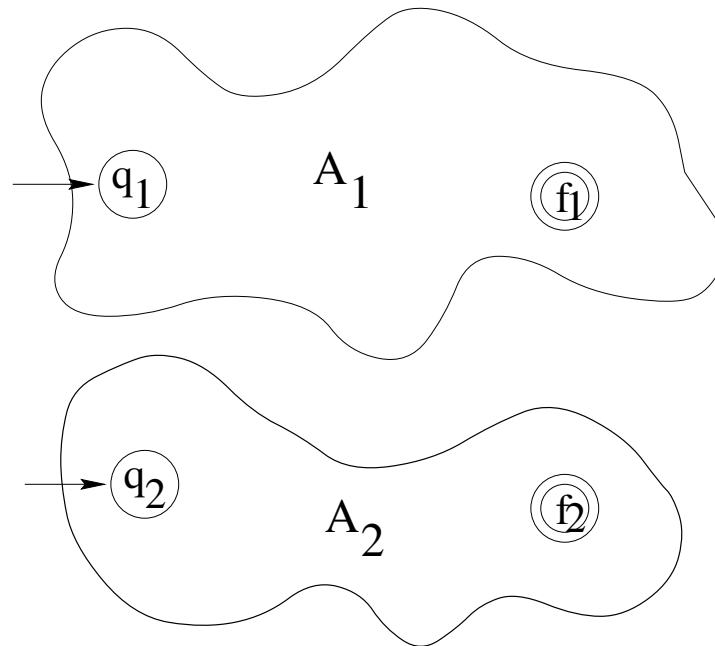
$r = \varepsilon$



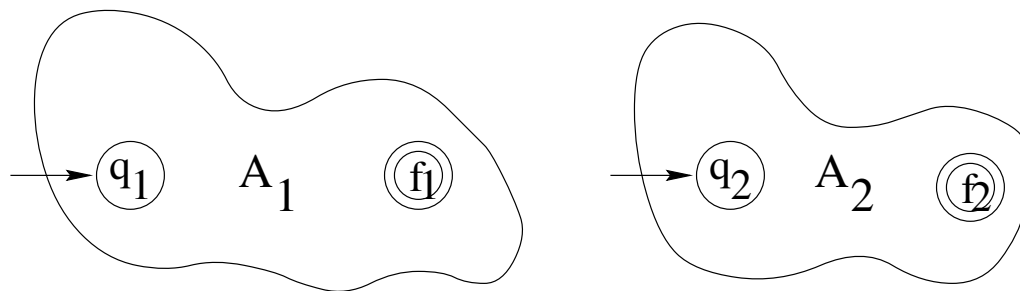
$r = a$

2. Inductive step. Assume we have ε -NFA A_1 and A_2 for regular expressions s and t , both with a single final state.

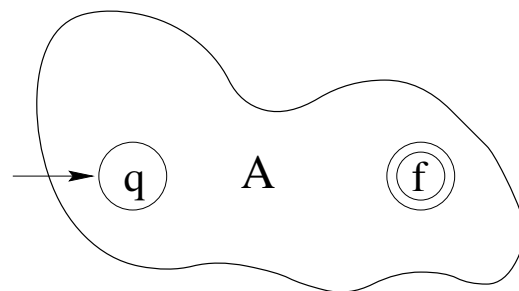
This is how we construct an ε -NFA for the union $s + t$:



This is an ε -NFA for the concatenation st :



Finally, let A accept s . This is an ε -NFA for the Kleene star s^* :



All constructions above work, and the resulting ε -NFA have exactly one final state. Using these constructions we can build an ε -NFA for any given regular expression. This completes the proof of (A).

Example. Let us construct an ε -NFA for regular expression

$$(abb^* + a)^*.$$

(B) DFA $\longrightarrow$ Regular expression

Let

$$A = (Q, \Sigma, \delta, q_0, F)$$

be the given DFA. States can be renamed as integers from 1 to n :

$$Q = \{1, 2, \dots, n\}.$$

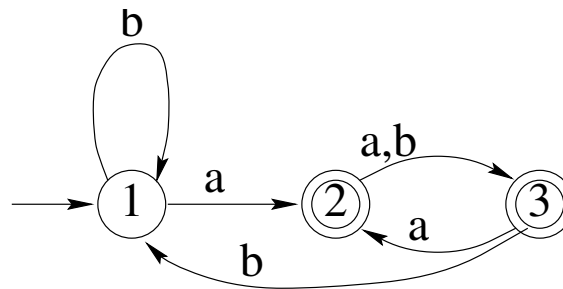
In this construction we build regular expressions for the following languages R_{ij}^k :

$$R_{ij}^k = \{w \mid w \text{ takes the automaton from state } i \text{ to state } j \text{ without} \\ \text{passing through any states greater than } k.\}.$$

"Passing through" a state means that the node is along the computation path, excluding the starting and ending points.

$R_{ij}^k = \{w \mid w \text{ takes the automaton from state } i \text{ to state } j \text{ without passing through any states greater than } k.\}$.

Example. In



the word $bbab$ belongs to R_{33}^2 but the word $baaab$ does not.

Also the word $bbaba$ is not in R_{33}^2 since its computation path does not end in state 3.

We construct regular expressions r_{ij}^k for the languages R_{ij}^k . We start with $k = 0$ and then move up to larger values of k .

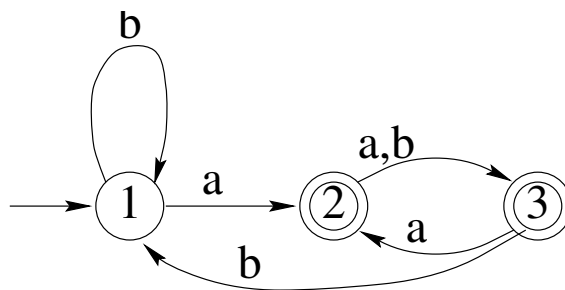
We construct regular expressions r_{ij}^k for the languages R_{ij}^k . We start with $k = 0$ and then move up to larger values of k .

1) $k = 0$. Language R_{ij}^0 contains all words that take the automaton from state i into state j without going through **any** states at all. Only possible input words are ε (if $i = j$) and single letters a (if $\delta(i, a) = j$).

If $a_1, a_2, \dots, a_p$ are the input letters with transitions from state i to j then R_{ij}^0 has regular expression

- $a_1 + a_2 + \dots + a_p$ if $i \neq j$ and $p \geq 1$,
- $\emptyset$ if $i \neq j$ and $p = 0$,
- $\varepsilon + a_1 + a_2 + \dots + a_p$ if $i = j$ and $p \geq 1$,
- ε if $i = j$ and $p = 0$.

Example. In



we have

$$r_{11}^0 =$$

$$r_{12}^0 =$$

$$r_{13}^0 =$$

$$r_{21}^0 =$$

$$r_{22}^0 =$$

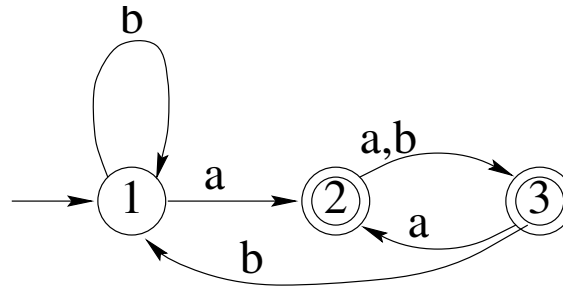
$$r_{23}^0 =$$

$$r_{31}^0 =$$

$$r_{32}^0 =$$

$$r_{33}^0 =$$

Example. In



we have

$$r_{11}^0 = \varepsilon + b$$

$$r_{12}^0 = a$$

$$r_{13}^0 = \emptyset$$

$$r_{21}^0 = \emptyset$$

$$r_{22}^0 = \varepsilon$$

$$r_{23}^0 = a + b$$

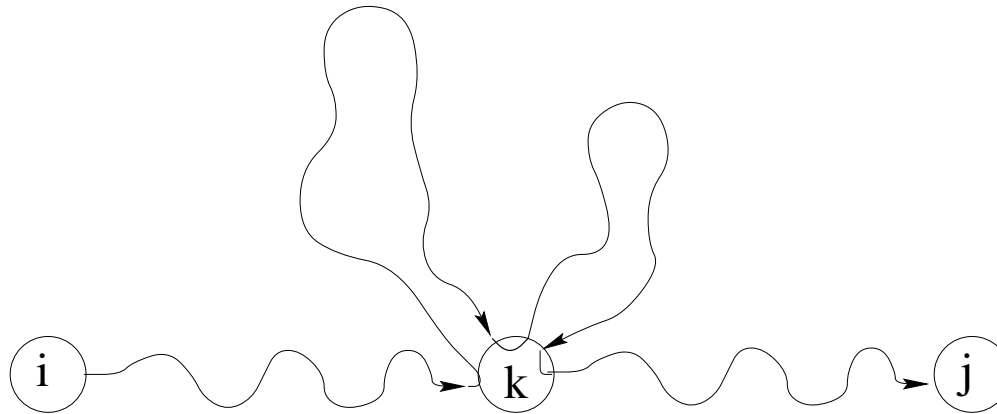
$$r_{31}^0 = b$$

$$r_{32}^0 = a$$

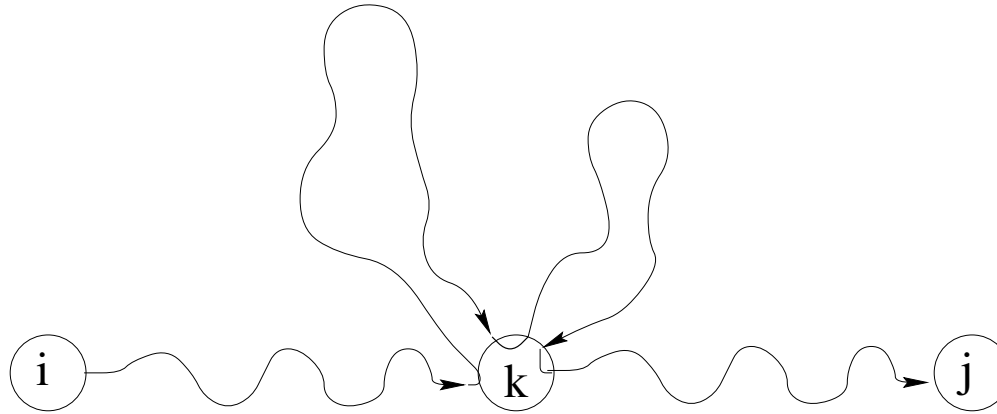
$$r_{33}^0 = \varepsilon$$

2) $k > 0$, and assume we have constructed regular expressions for all R_{ij}^{k-1} .

Consider an arbitrary computation path from state i into state j that only goes through states $\{1, 2, \dots, k\}$. Let us cut the path into segments at points where it goes through state k :

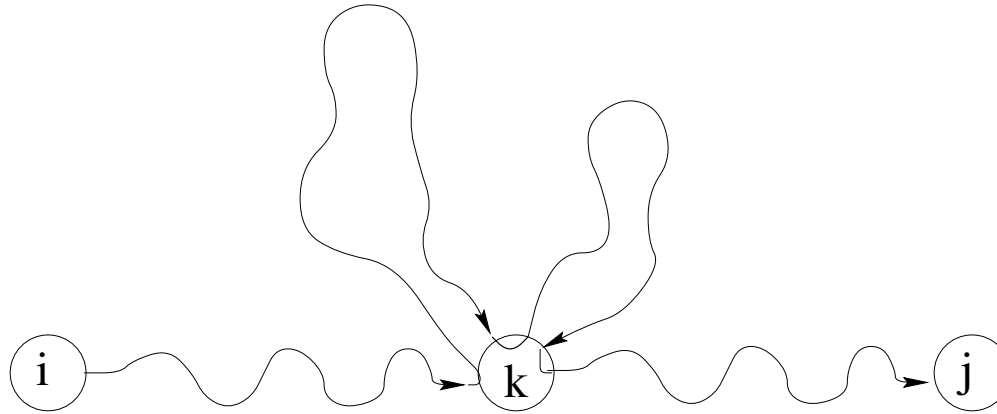


All segments only go through states $\{1, 2, \dots, k-1\}$. The first segment is in the set R_{ik}^{k-1} , and the last segment belongs to R_{kj}^{k-1} . All middle segments start and end in state k , so they belong to R_{kk}^{k-1} . Number of middle segments can be arbitrary.



So words of R_{ij}^k that visit k at least once form the language $R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}^{k-1}$.

We also must include words that do not visit state k even once, i.e., R_{ij}^{k-1} .



So words of R_{ij}^k that visit k at least once form the language $R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}^{k-1}$.

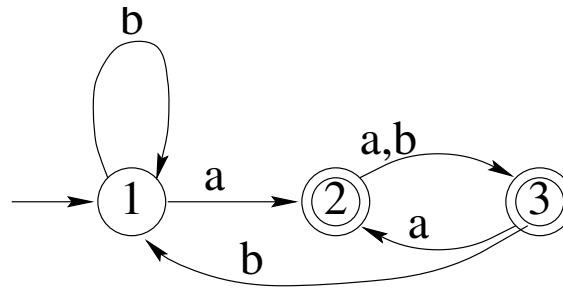
We also must include words that do not visit state k even once, i.e., R_{ij}^{k-1} .

In total: $R_{ij}^k = R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}^{k-1} \cup R_{ij}^{k-1}$

So the language R_{ij}^k is defined by the regular expression

$$r_{ij}^k = r_{ik}^{k-1} (r_{kk}^{k-1})^* r_{kj}^{k-1} + r_{ij}^{k-1}$$

Example.



$$r_{11}^0 = \varepsilon + b$$

$$r_{11}^1 =$$

$$r_{11}^2 =$$

$$r_{11}^3 =$$

$$r_{12}^0 = a$$

$$r_{12}^1 =$$

$$r_{12}^2 =$$

$$r_{12}^3 =$$

$$r_{13}^0 = \emptyset$$

$$r_{13}^1 =$$

$$r_{13}^2 =$$

$$r_{13}^3 =$$

$$r_{21}^0 = \emptyset$$

$$r_{21}^1 =$$

$$r_{21}^2 =$$

$$r_{21}^3 =$$

$$r_{22}^0 = \varepsilon$$

$$r_{22}^1 =$$

$$r_{22}^2 =$$

$$r_{22}^3 =$$

$$r_{23}^0 = a + b$$

$$r_{23}^1 =$$

$$r_{23}^2 =$$

$$r_{23}^3 =$$

$$r_{31}^0 = b$$

$$r_{31}^1 =$$

$$r_{31}^2 =$$

$$r_{31}^3 =$$

$$r_{32}^0 = a$$

$$r_{32}^1 =$$

$$r_{32}^2 =$$

$$r_{32}^3 =$$

$$r_{33}^0 = \varepsilon$$

$$r_{33}^1 =$$

$$r_{33}^2 =$$

$$r_{33}^3 =$$

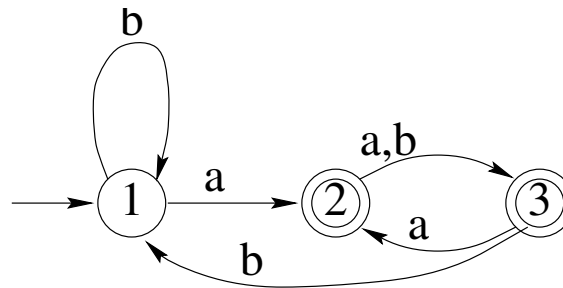
The regular expression r_{ij}^n represents all strings that take the automaton from state i to state j through any states. If $i = 1$ is the initial state then r_{1j}^n represents strings whose computation paths finish in state j .

If states $j_1, j_2, \dots, j_f$ are the final states of the DFA then the language recognized by the automaton is represented by the expression

$$r_{1j_1}^n + r_{1j_2}^n + \cdots + r_{1j_f}^n.$$

These are all words that take the machine from the initial state to some final state, using any states whatsoever on the way.

Example.



The language recognized by this DFA is given by regular expression

$$r_{12}^3 + r_{13}^3 =$$

The construction provides huge regular expressions even for small automata. It is a good idea to simplify the expressions r_{ij}^k as you go along. Following simplifications are especially useful.

For any regular expressions r , s and t , both sides of following simplification rules describe the same language:

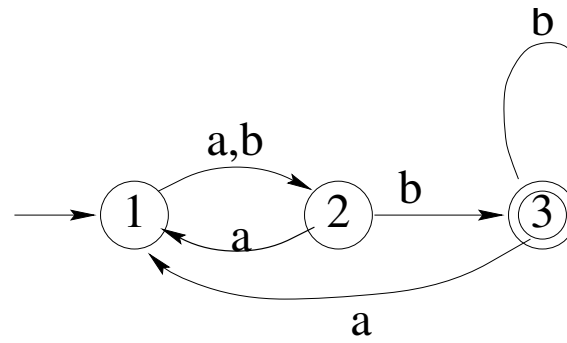
$$\begin{array}{ll}
 \varepsilon^* & \longrightarrow \varepsilon \\
 \varepsilon r, r\varepsilon & \longrightarrow r \\
 (r + \varepsilon)^* & \longrightarrow r^* \\
 \emptyset r, r\emptyset & \longrightarrow \emptyset \\
 \emptyset + r, r + \emptyset & \longrightarrow r \\
 r(s + t) & \longrightarrow rs + rt \\
 (s + t)r & \longrightarrow sr + tr \\
 (r + \varepsilon)r^*, r^*(r + \varepsilon) & \longrightarrow r^*
 \end{array}$$

Also, one may only construct only those expressions r_{ij}^k that are actually needed.

For example, in our sample DFA we only need two regular expressions r_{12}^3 and r_{13}^3 at level $k = 3$.

At level $k = 2$ we only need to construct expressions that are used at level 3: In our case r_{12}^2 , r_{13}^2 , r_{33}^2 and r_{32}^2 . And so on.

Example. Let us find a regular expression equivalent to the following DFA:



Another **language equation** based method to construct a regular expression from a DFA. Again, we number the states from 1 to n :

$$Q = \{1, 2, \dots, n\},$$

and assume that 1 is the initial state.

For every $i, j \in Q$, denote

$$L_i = \{w \in \Sigma^* \mid \delta(1, w) = i\}$$

and

$$K_{ij} = \{a \in \Sigma \mid \delta(i, a) = j\}.$$

Then the following equalities of languages hold:

$$\begin{aligned} L_1 &= L_1 K_{11} \cup L_2 K_{21} \cup \dots \cup L_n K_{n1} \cup \{\varepsilon\}, \\ L_2 &= L_1 K_{12} \cup L_2 K_{22} \cup \dots \cup L_n K_{n2}, \\ &\vdots \\ L_n &= L_1 K_{1n} \cup L_2 K_{2n} \cup \dots \cup L_n K_{nn}. \end{aligned}$$

In each equation the language on the left and on the right are identical. The languages K_{ij} can be read directly from the given DFA – they are a representation of the transitions. The languages L_i are unknowns that we want to solve from the system of equations.

$$\begin{aligned}
L_1 &= L_1K_{11} \cup L_2K_{21} \cup \dots \cup L_nK_{n1} \cup \{\varepsilon\}, \\
L_2 &= L_1K_{12} \cup L_2K_{22} \cup \dots \cup L_nK_{n2}, \\
&\vdots \\
L_n &= L_1K_{1n} \cup L_2K_{2n} \cup \dots \cup L_nK_{nn}.
\end{aligned}$$

Note that the languages K_{ij} do not contain the empty word ε . It turns out for such K_{ij} the languages L_i are uniquely determined from the system of equations.

The following lemma states this for the case $n = 1$:

Lemma. Let $K, L \subseteq \Sigma^*$ be languages, and assume that $\varepsilon \notin K$. Then $X = LK^*$ is the unique language for which the equality $X = XK \cup L$ holds.

Proof.

$$\begin{aligned}
L_1 &= L_1K_{11} \cup L_2K_{21} \cup \dots \cup L_nK_{n1} \cup \{\varepsilon\}, \\
L_2 &= L_1K_{12} \cup L_2K_{22} \cup \dots \cup L_nK_{n2}, \\
&\vdots \\
L_n &= L_1K_{1n} \cup L_2K_{2n} \cup \dots \cup L_nK_{nn}.
\end{aligned}$$

Using the Lemma on the first equation one may solve L_1 , substitute the solution to the other equations, solve L_2 from the second equation, substitute it, and so on.

Example. Let us use the method of language equations to find a regular expression for the language recognized by

