Classes and conversions

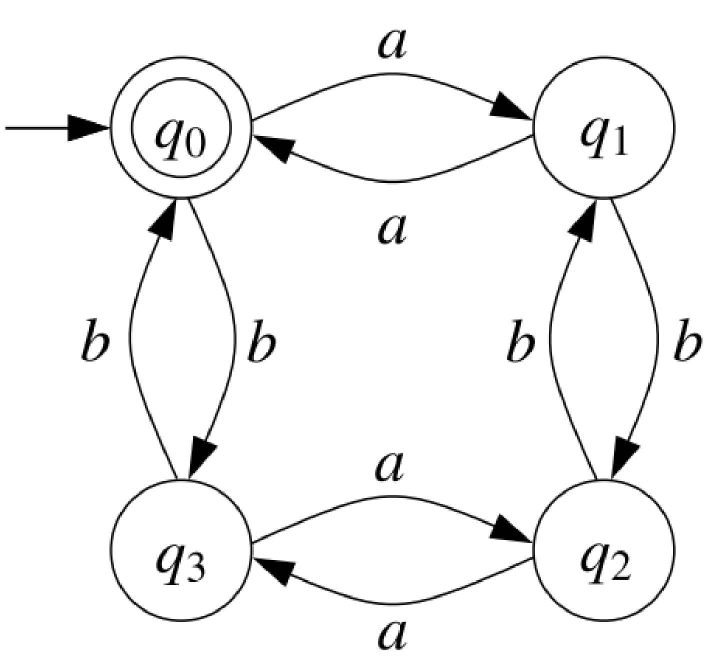
Regular expressions

- Syntax: $r ::= \emptyset \mid \epsilon \mid a \mid r_1 r_2 \mid r_1 + r_2 \mid r^*$
- Semantics: The language $L(r)$ of a regular expression r is inductively defined as follows:
  - $L(\emptyset) = \emptyset$, $L(\epsilon) = \{\epsilon\}$, $L(a) = \{a\}$
  - $L(r_1 r_2) = L(r_1)L(r_2)$
    where $L_1 L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$
  - $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
  - $L(r^*) = \bigcup_{i \geq 0} L^i$
    where $L^0 = \{\epsilon\}$ and $L^{i+1} = L^i L$

Deterministic finite automata (DFA)

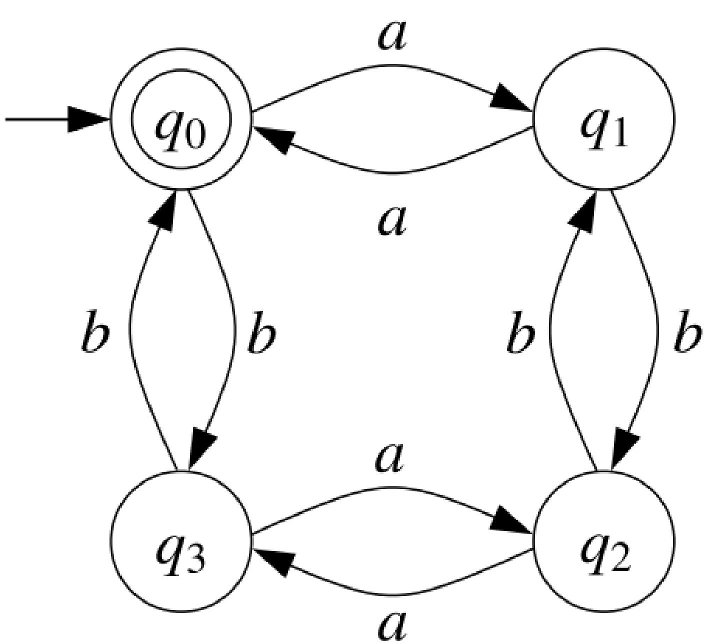
A deterministic finite automaton is a tuple $A = (Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite, nonempty set of states
- Σ is a nonempty, finite set of letters, called an alphabet
- $\delta: Q \times \Sigma \to Q$ is the transition function
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is the set of final states



Run of a DFA on a word

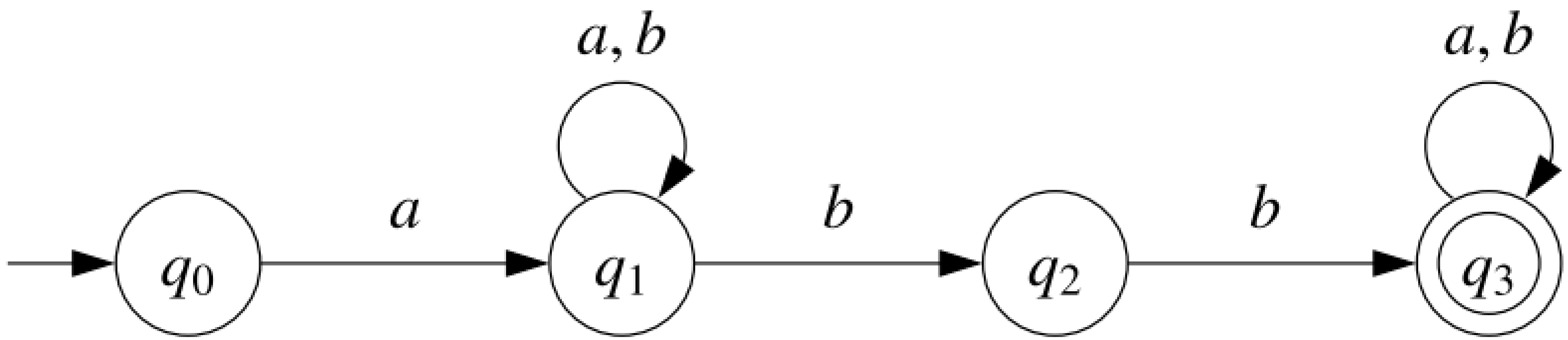
- $q \xrightarrow{a} q'$ denotes $\delta(q, a) = q'$
- The run of a DFA on a word $a_1 a_2 \ldots a_n \in \Sigma^*$ is the unique sequence $q_0 q_1 \ldots q_n$ of states such that
$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots q_{n-1} \xrightarrow{a_n} q_n$$
- A DFA accepts a word iff its run on it ends in a final state. We say the run is accepting.
- A DFA over an alphabet Σ recognizes a language $L \subseteq \Sigma^*$ if it accepts every word of L and no other. The language recognized by a DFA A is denoted $L(A)$.



Nondeterministic finite automata (NFA)

A nondeterministic automaton is a tuple $A = (Q, \Sigma, \delta, Q_0, F)$ where

- Q, Σ, F are as for DFAs
- $\delta: Q \times \Sigma \to 2^Q$ is the transition function
- $Q_0 \in Q$ is the set of initial states

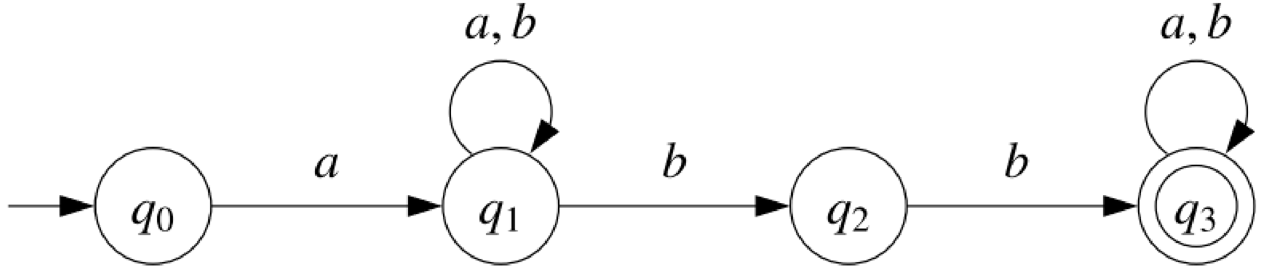


Runs of an NFA on a word

- A run of an NFA on a word $a_1 a_2 \dots a_n \in \Sigma^*$ is a sequence $q_0 q_1 \dots q_n$ of states such that $q_0 \in Q_0$ and

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots q_{n-1} \xrightarrow{a_n} q_n$$

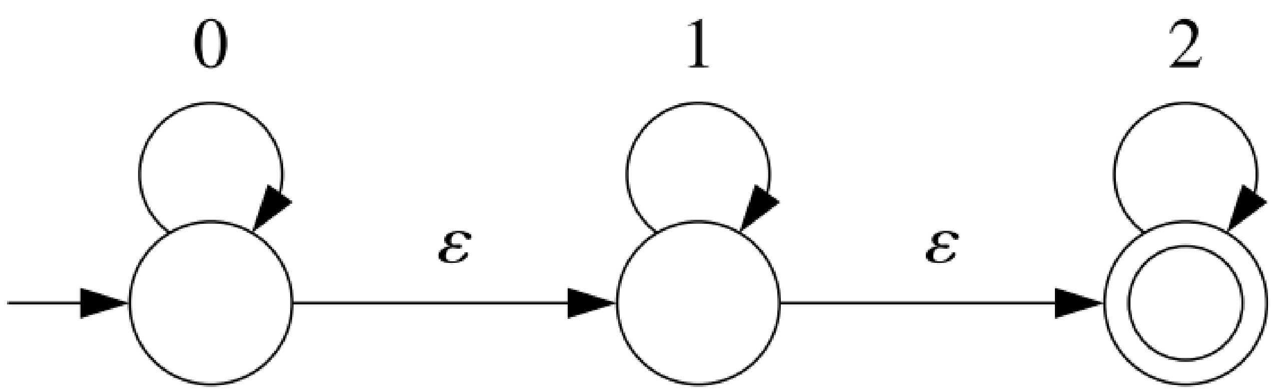
- An NFA can have 0, 1, or more runs on the same word (but only finitely many).
- An NFA accepts a word iff at least one of its runs on it is accepting.



Nondeterministic finite automata with ϵ-transitions (NFAϵ)

A nondeterministic automaton with ϵ-transitions is a tuple $A = (Q, \Sigma, \delta, Q_0, F)$ where

- Q, Σ, Q_0, F are as for NFAs
- $\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$ is the transition function



Runs of an NFAϵ on a word

- A run of an NFAϵ on a word $a_1 a_2 \dots a_n \in \Sigma^*$ is a sequence $q_0 \dots q_0' q_1 \dots q_1' q_2 \dots q_{n-1}' q_n \cdots q_n'$ of states such that $q_0 \in Q_0$ and

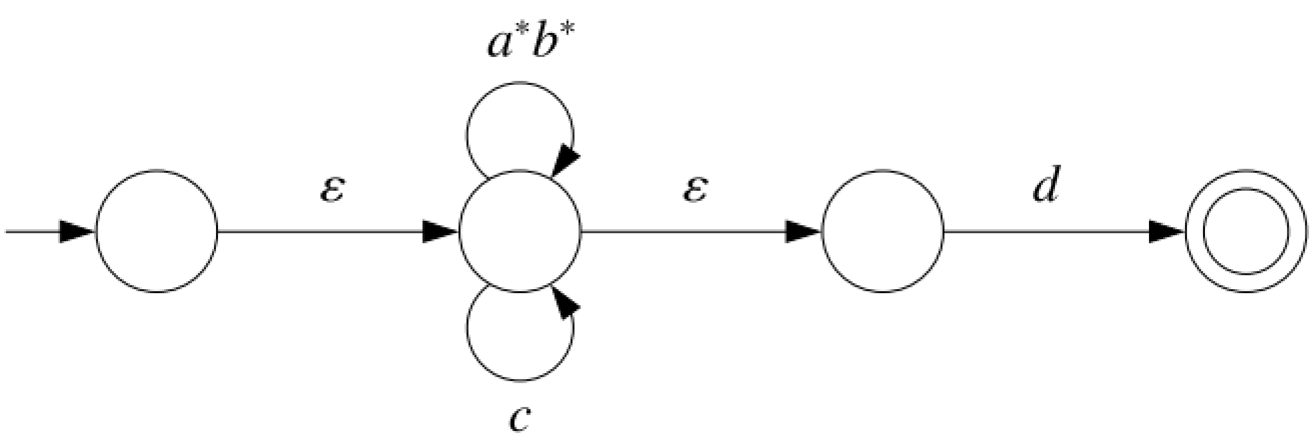
$$q_0 \xrightarrow{\epsilon} \cdots \xrightarrow{\epsilon} q_0' \xrightarrow{a_1} q_1 \xrightarrow{\epsilon} \cdots \xrightarrow{\epsilon} q_1' \xrightarrow{a_2} q_2 \cdots q_{n-1}' \xrightarrow{a_n} q_n \xrightarrow{\epsilon} \cdots \xrightarrow{\epsilon} q_n'$$

- An NFAϵ can have 0, 1, or more runs on the same word, even infinitely many.
- An NFAϵ accepts a word iff at least one of its runs on it is accepting.

Nondeterministic finite automata with regular expressions (NFAreg)

A nondeterministic automaton with regular expressions is a tuple $A = (Q, \Sigma, \delta, Q_0, F)$ where

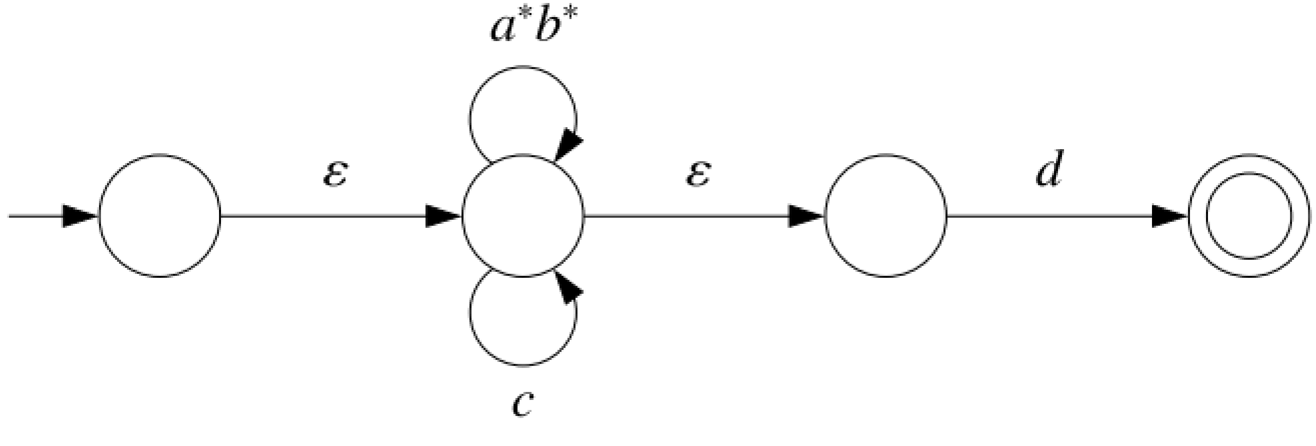
- Q, Σ, Q_0, F are as for NFAs
- $\delta: Q \times \text{Reg}(\Sigma) \to 2^Q$ is the transition function, where $\delta(q, r) = \emptyset$ is the case for all but finitely many pairs $(q, r) \in Q \times \text{Reg}(\Sigma)$



Language recognized by an NFAreg

An NFAreg accepts a word w if there are states $q_0, \dots, q_n$ and regular expressions $r_1, \dots, r_n$ such that

- $q_0 \in Q_0$, $q_n \in F$,
- $q_0 \xrightarrow{r_1} q_1 \xrightarrow{r_2} q_2 \cdots q_{n-1} \xrightarrow{r_n} q_n$, and
- $w \in L(r_1 r_2 \cdots r_n)$.

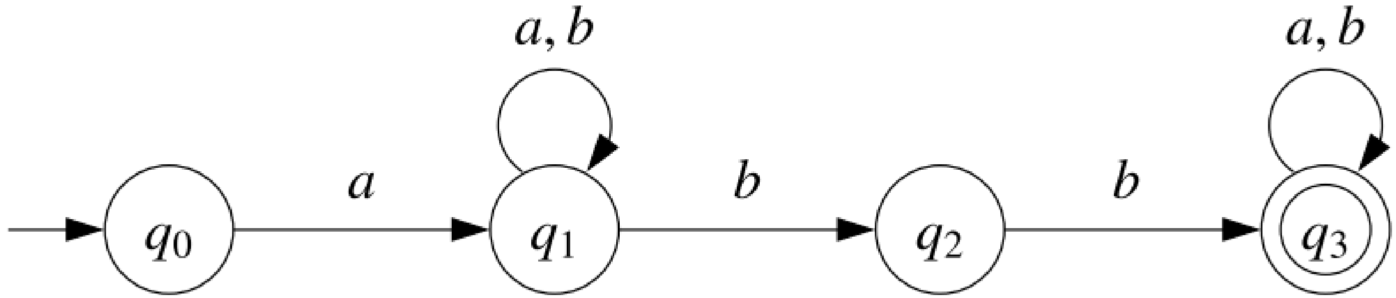


Normal form

- An automaton of any class is in normal form if every state is reachable by a path of transitions from some initial state.
- For every automaton there is an equivalent automaton in normal form.
- All algorithms in this course assume that automata inputs are in normal form, and guarantee that the output is also in normal form.

Conversions

NFA to DFA



The powerset construction

NFAtoDFA(A)

Input: NFA $A = (Q, \Sigma, \delta, Q_0, F)$

Output: DFA $B = (\mathcal{Q}, \Sigma, \Delta, q_0, \mathcal{F})$ with $L(B) = L(A)$

$\mathcal{Q}, \Delta, \mathcal{F} \leftarrow \emptyset$; $q_0 \leftarrow Q_0$

$\mathcal{W} = \{Q_0\}$

while $\mathcal{W} \neq \emptyset$ **do**

  pick Q' **from** $\mathcal{W}$

  add Q' **to** $\mathcal{Q}$

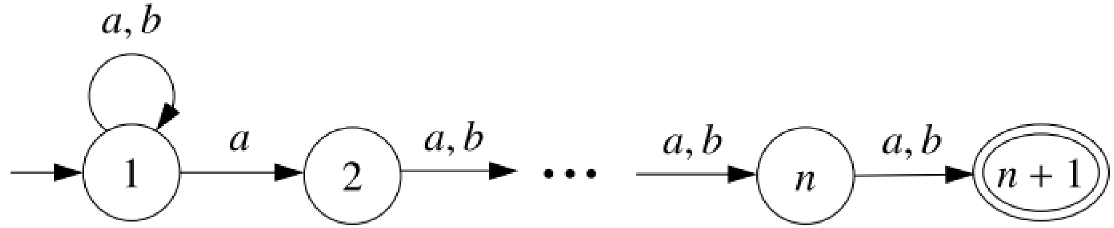
  if $Q' \cap F \neq \emptyset$ **then add** Q' **to** $\mathcal{F}$

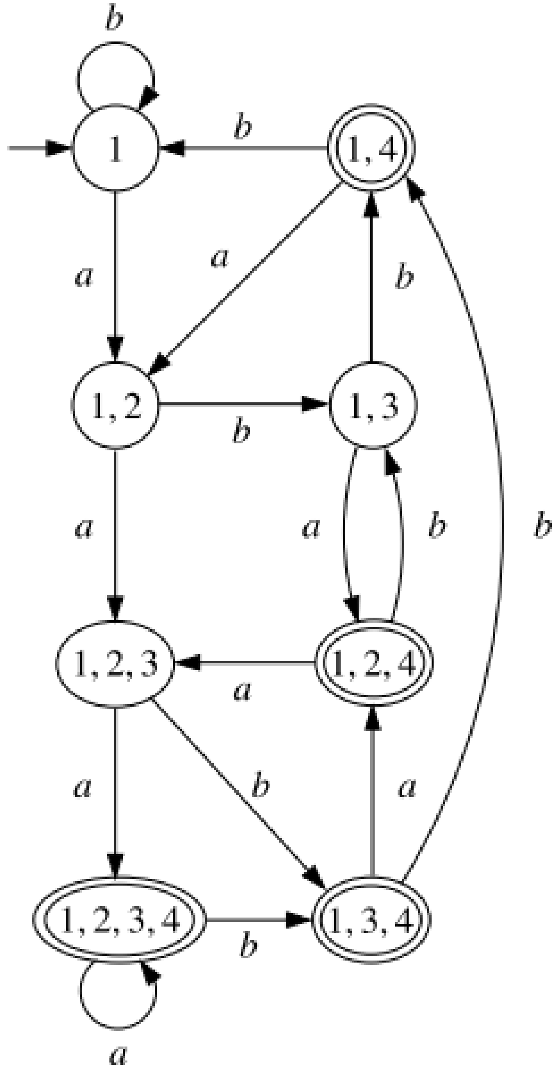
  for all $a \in \Sigma$ **do**

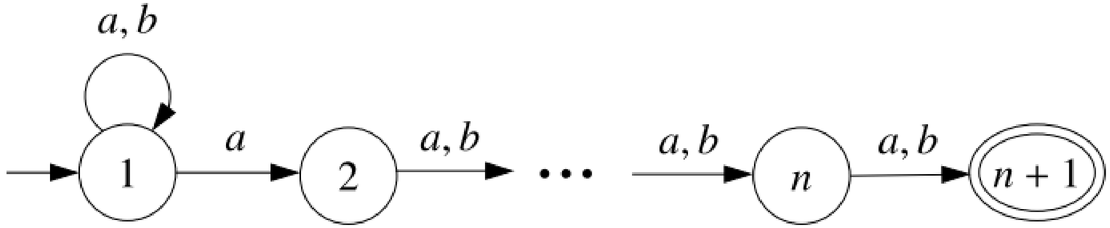
    $Q'' \leftarrow \bigcup_{q \in Q'} \delta(q, a)$

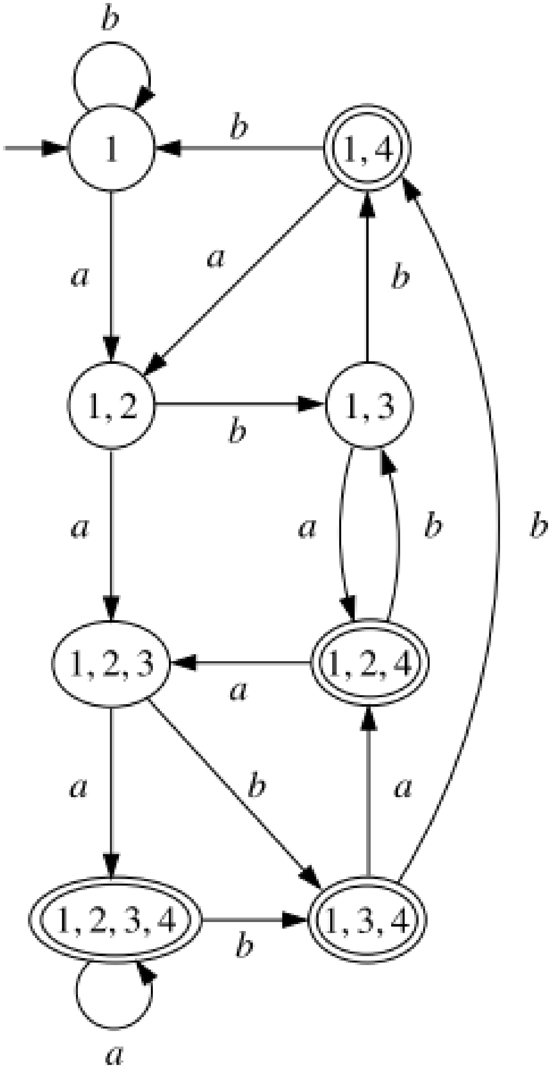
    if $Q'' \notin \mathcal{Q}$ **then add** Q'' **to** $\mathcal{W}$

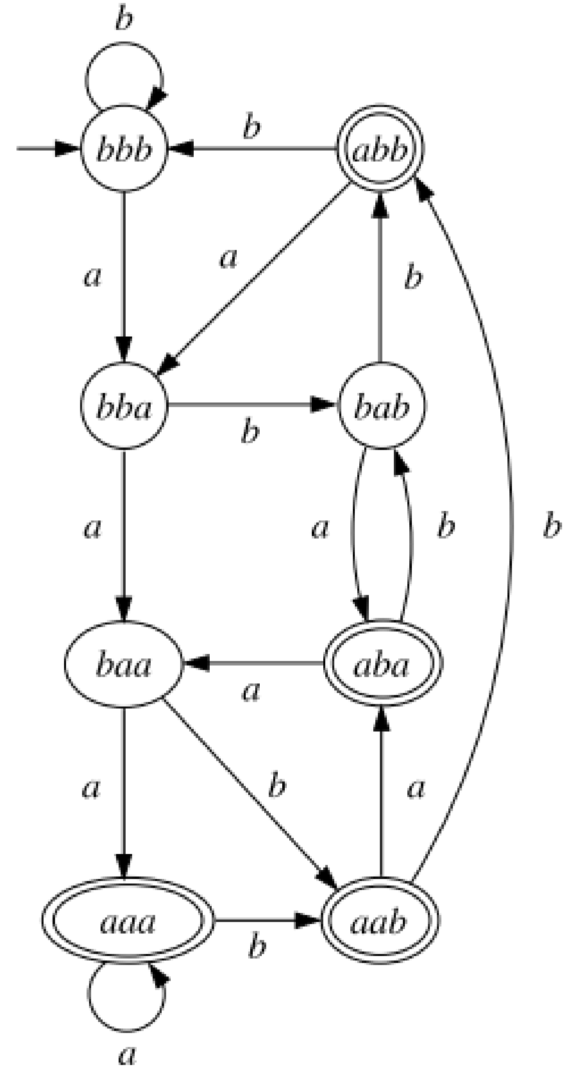
    add (Q', a, Q'') **to** Δ


a, b
1
a
2
a, b
a, b
n
a, b
n + 1


b
1
b
1, 4
a
a
b
1, 2
b
1, 3
a
a
b
b
1, 2, 3
a
1, 2, 4
a
b
a
1, 2, 3, 4
b
1, 3, 4
a


a, b
1
a
2
a, b
a, b
n
a, b
n + 1


1
1, 2
1, 3
1, 4
1, 2, 3
1, 2, 4
1, 2, 3, 4
1, 3, 4
a
b


bbb
abb
bba
bab
baa
aba
aaa
aab
a
b

Let L_n be the language of the NFA with $n + 1$ states.

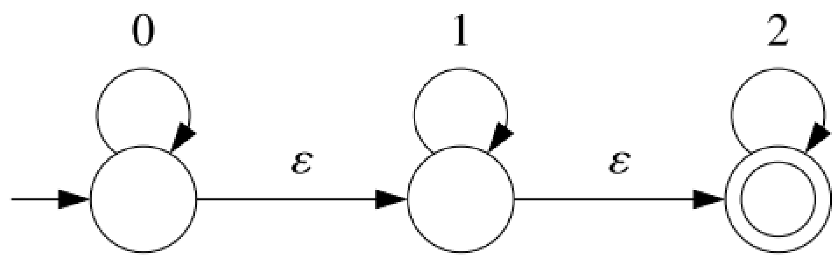
Proposition. Every DFA for L_n has at least 2^n states.

Proof: Assume some DFA for L_n has fewer states.

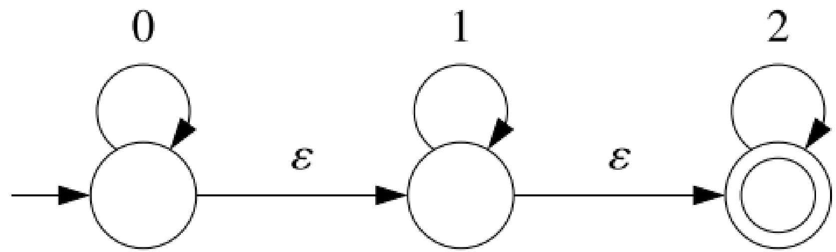
Then two different words of length n lead to the same state. Let the words be uav and ubw.

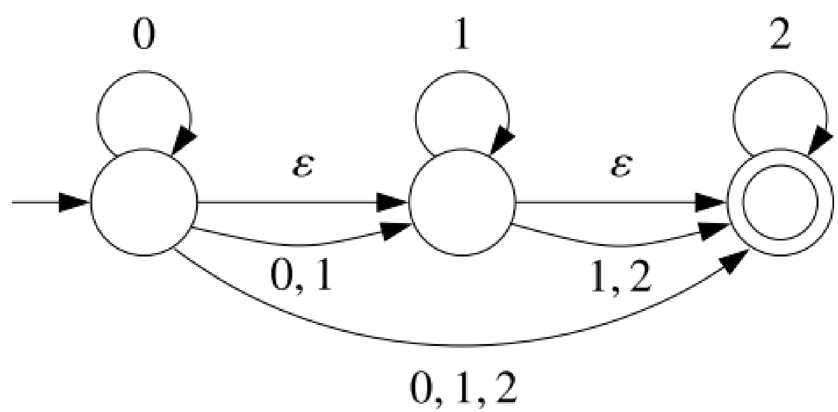
Then $uavu$ and $ubwu$ lead to the same state too, but only $uavu$ is accepted. Contradiction.

NFAϵ to NFA



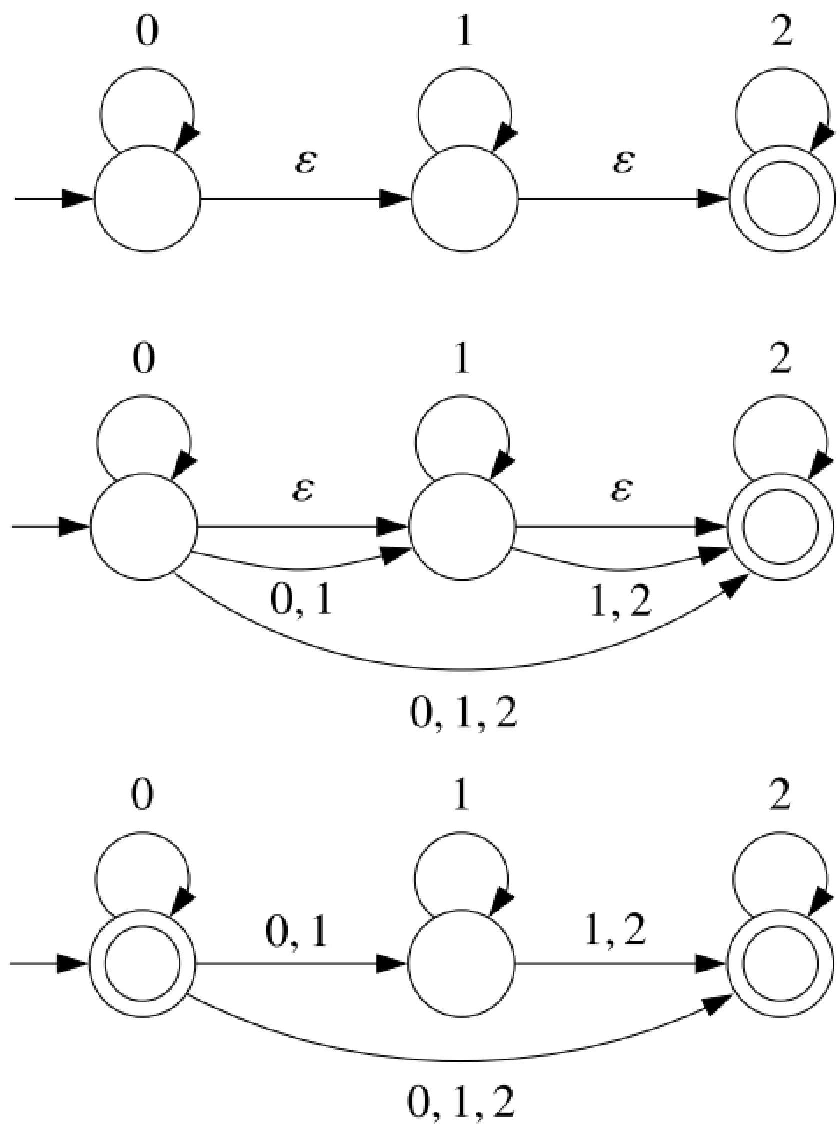
NFAϵ to NFA





Saturation

NFAϵ to NFA



Saturation

Check of the initial state + ϵ-removal

A one-pass algorithm

NFAεtoNFA(A)

Input: NFA-ε $A = (Q, \Sigma, \delta, Q_0, F)$

Output: NFA $B = (Q', \Sigma, \delta', Q'_0, F')$ with $L(B) = L(A)$

$Q'_0 \leftarrow Q_0$
$Q' \leftarrow Q_0$; $\delta' \leftarrow \emptyset$; $F' \leftarrow F \cap Q_0$
$\delta'' \leftarrow \emptyset$; $W \leftarrow \{(q, \alpha, q') \in \delta \mid q \in Q_0\}$
while $W \neq \emptyset$ **do**
    pick (q_1, α, q_2) **from** W
    if $\alpha \neq \varepsilon$ **then**
        add q_2 **to** Q'; **add** (q_1, α, q_2) **to** δ'; **if** $q_2 \in F$ **then add** q_2 **to** F'
        for all $q_3 \in \delta(q_2, \varepsilon)$ **do**
            if $(q_1, \alpha, q_3) \notin \delta'$ **then add** (q_1, α, q_3) **to** W
        for all $a \in \Sigma, q_3 \in \delta(q_2, a)$ **do**
            if $(q_2, a, q_3) \notin \delta'$ **then add** (q_2, a, q_3) **to** W
    else $/ * \alpha = \varepsilon * /$
        add (q_1, α, q_2) **to** δ''; **if** $q_2 \in F$ **then add** q_1 **to** F'
        for all $\beta \in \Sigma \cup \{\varepsilon\}, q_3 \in \delta(q_2, \beta)$ **do**
            if $(q_1, \beta, q_3) \notin \delta' \cup \delta''$ **then add** (q_1, β, q_3) **to** W

Correctness

Proposition. Let A be an NFAϵ and let $B = \mathrm{NFA\epsilon toNFA}(A)$. Then B is an NFA and $L(A) = L(B)$.

Proof.

- Termination. Every transition that leaves W is never added to W again, and each iteration of the while loop removes one transition from W.
- B is an NFA. Easy.
- $L(B) \subseteq L(A)$.
  - Check that every transition added by the algorithm is a transition of A or a shortcut.
  - Check that the algorithm only adds initial states as final, and only if A has an ϵ-path from them to a final state.
    Invariant: At line 13, $q_1 \in Q_0$. Proof by induction, observing that the algorithm only adds ϵ-transitions to W at line 15.

Correctness

- $L(A) \subseteq L(B)$

If $\epsilon \in L(A)$ then $\epsilon \in L(B)$

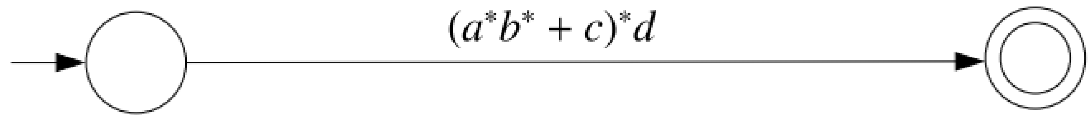
$$q_0 \xrightarrow{\epsilon} q_1 \xrightarrow{\epsilon} q_2 \xrightarrow{\epsilon} q_3 \xrightarrow{\epsilon} q_4$$

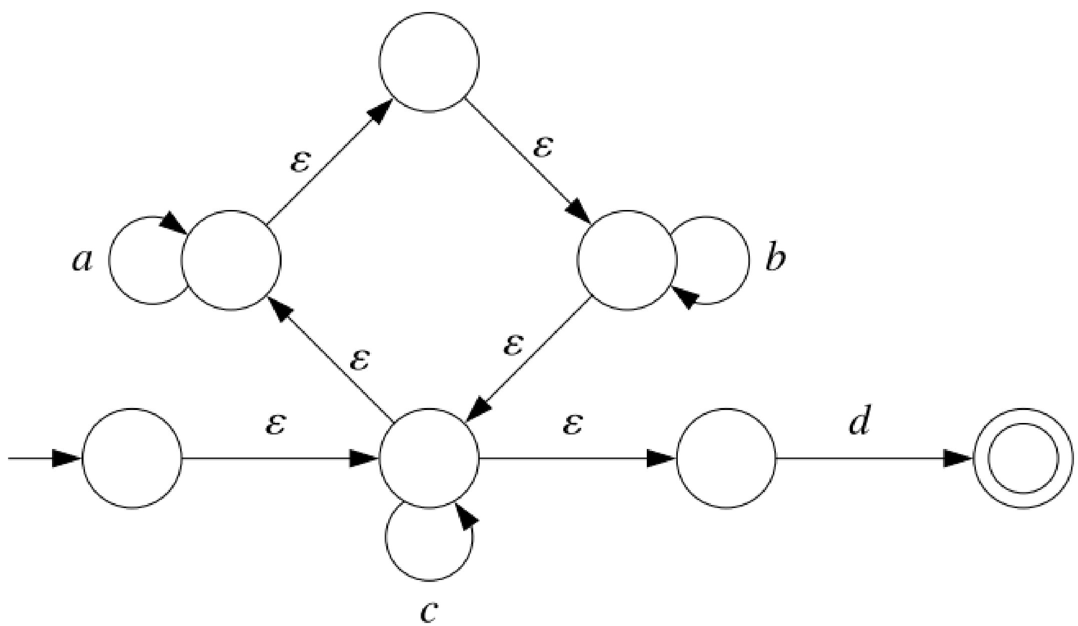
If $w \neq \epsilon$ and $w \in L(A)$ then $w \in L(B)$

$$q_0 \xrightarrow{\epsilon} q_1 \xrightarrow{\epsilon} q_2 \xrightarrow{a_1} q_3 \xrightarrow{\epsilon} q_4 \xrightarrow{\epsilon} q_5 \xrightarrow{a_2} q_5 \xrightarrow{\epsilon} q_6$$

Regular expressions to NFAϵ

$(a^*b^* + c)^*d$



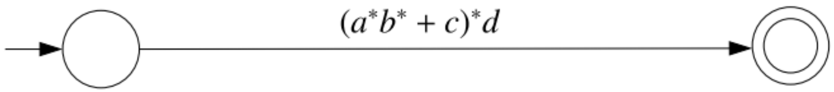


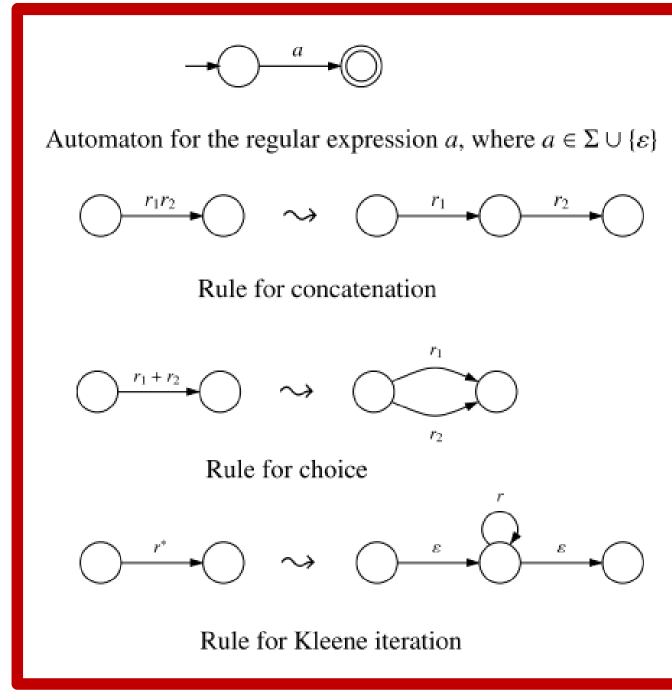
Regular expressions to NFAϵ

- Preprocessing: Convert the regular expression into another one which is either equal to ∅, or does not contain any occurrence of ∅.
- Use the following rewrite rules:

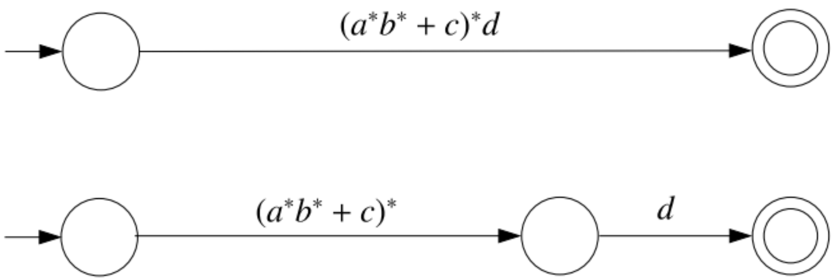
$$
\begin{array}{rcl}
\emptyset \cdot r & \rightsquigarrow & \emptyset \\
r + \emptyset & \rightsquigarrow & r \\
\emptyset^* & \rightsquigarrow & \varepsilon
\end{array}
\qquad\qquad
\begin{array}{rcl}
r \cdot \emptyset & \rightsquigarrow & \emptyset \\
\emptyset + r & \rightsquigarrow & r
\end{array}
$$

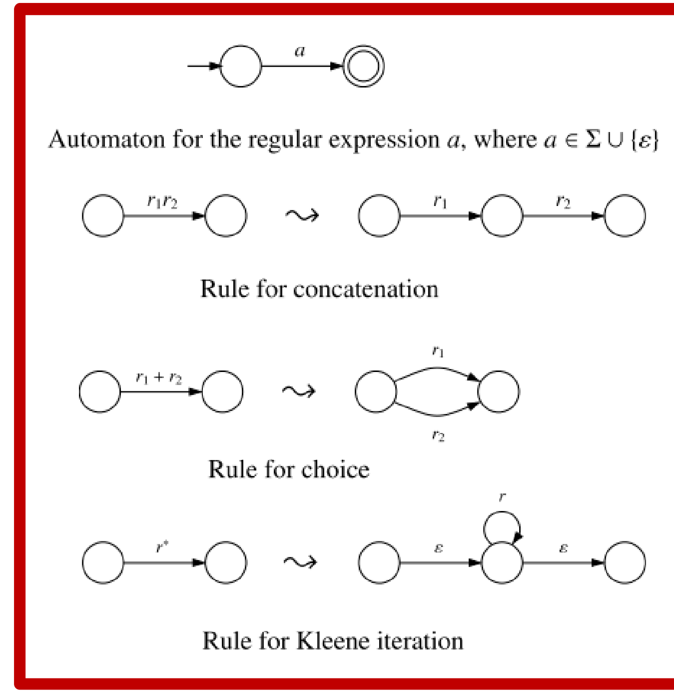
Regular expressions to NFAϵ



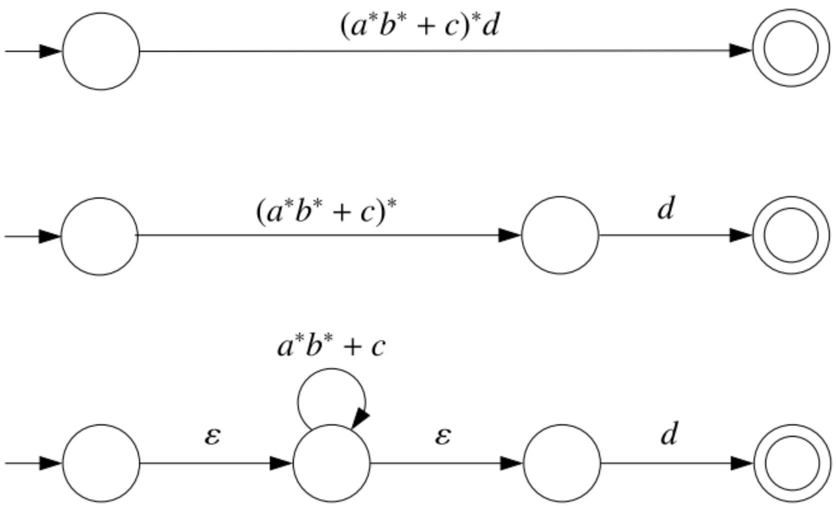


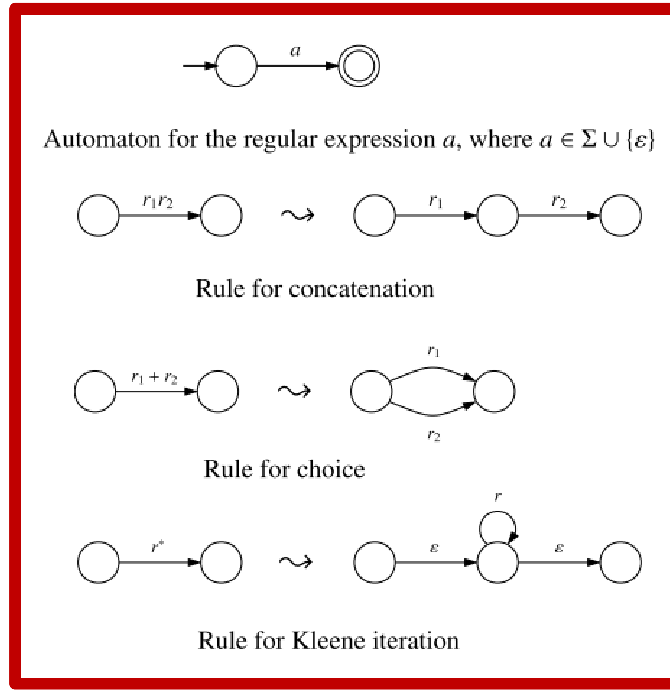
Regular expressions to NFAϵ



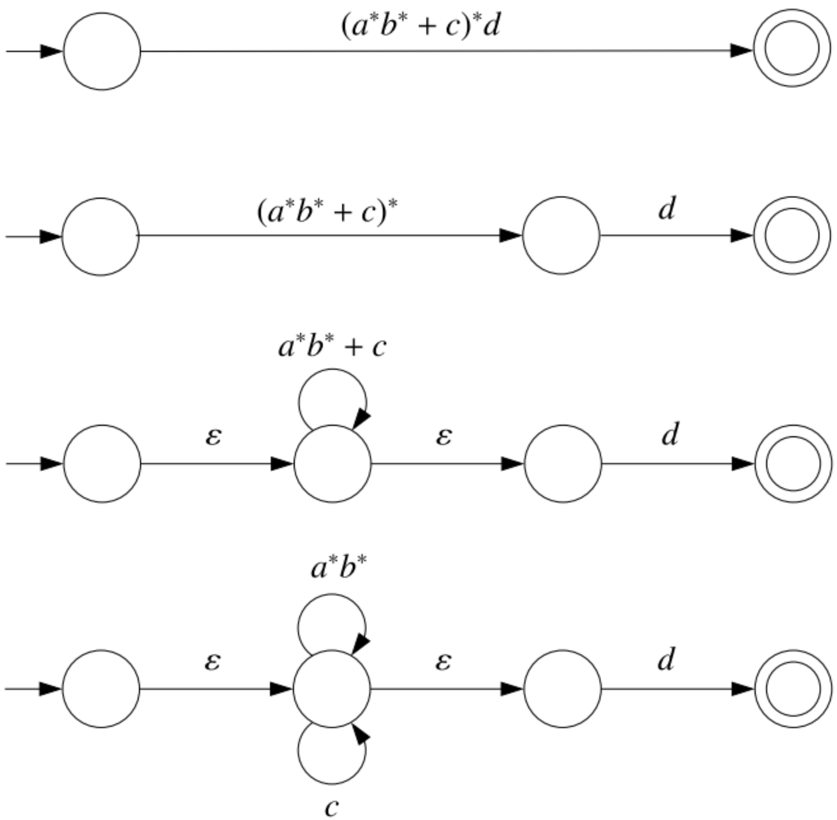


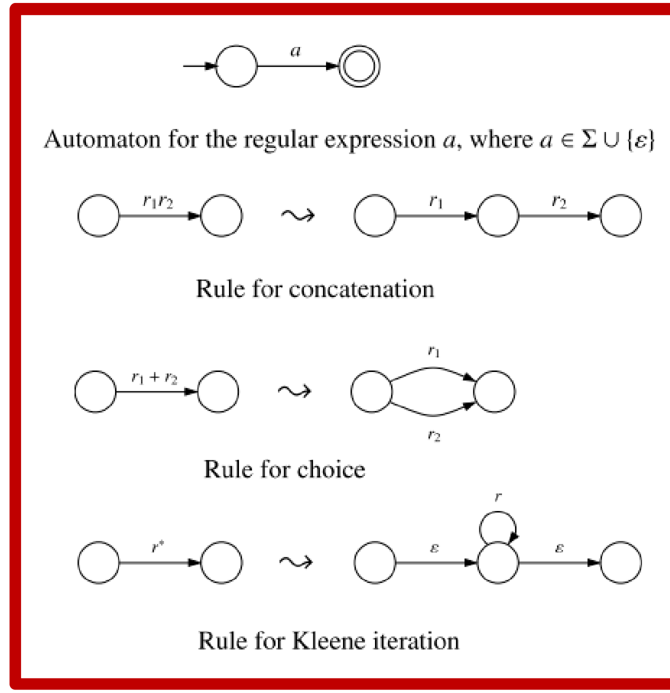
Regular expressions to NFAϵ



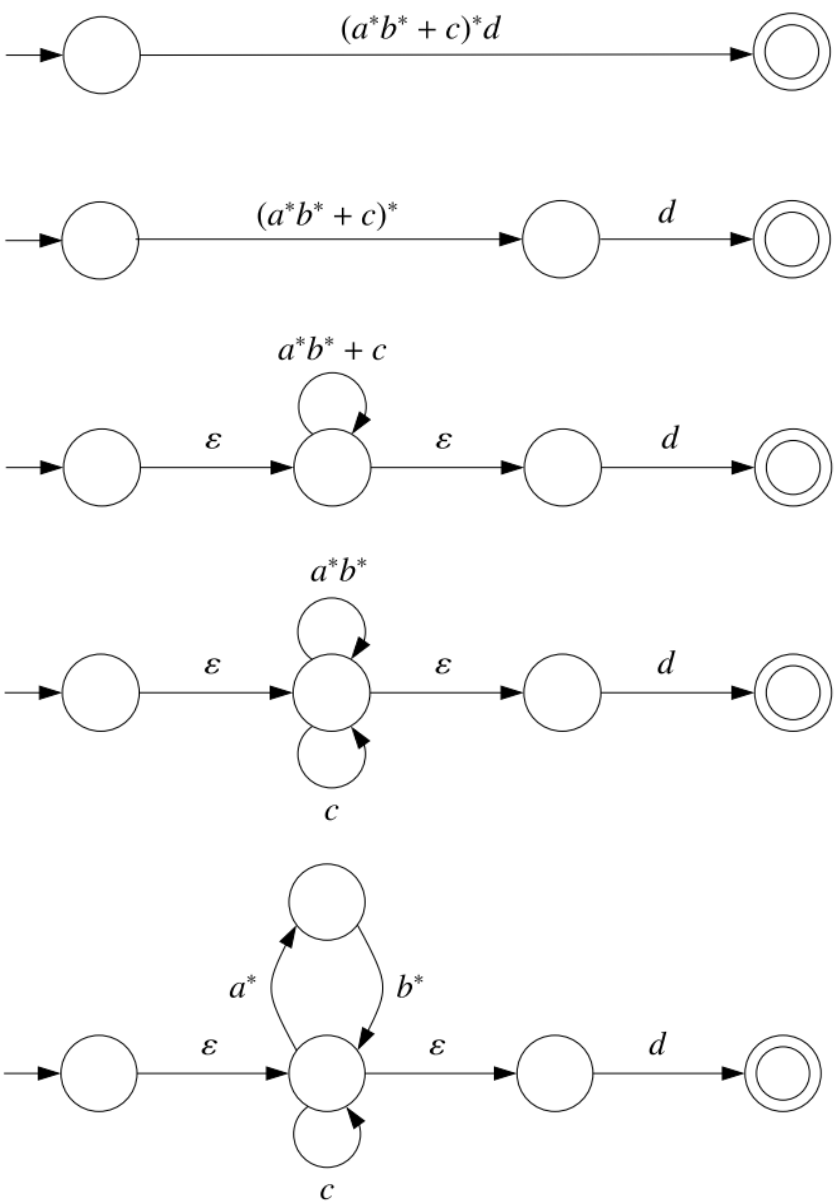


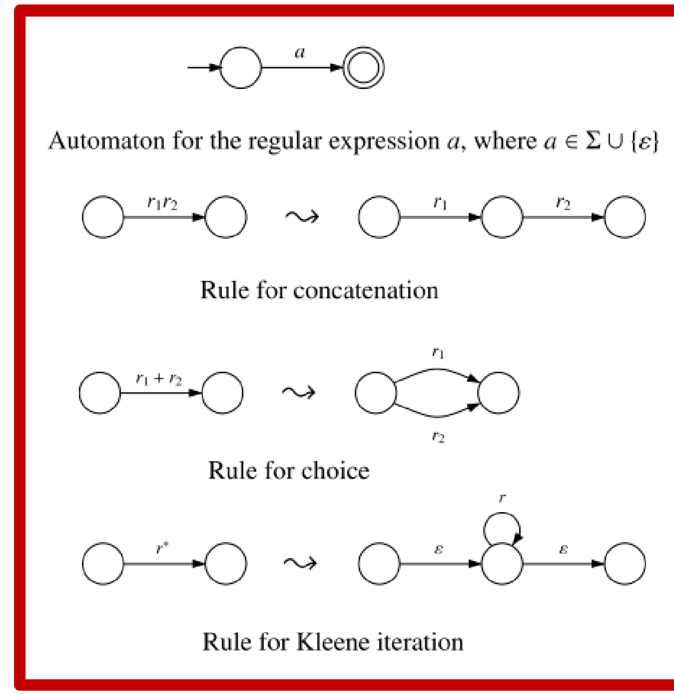
Regular expressions to NFAϵ



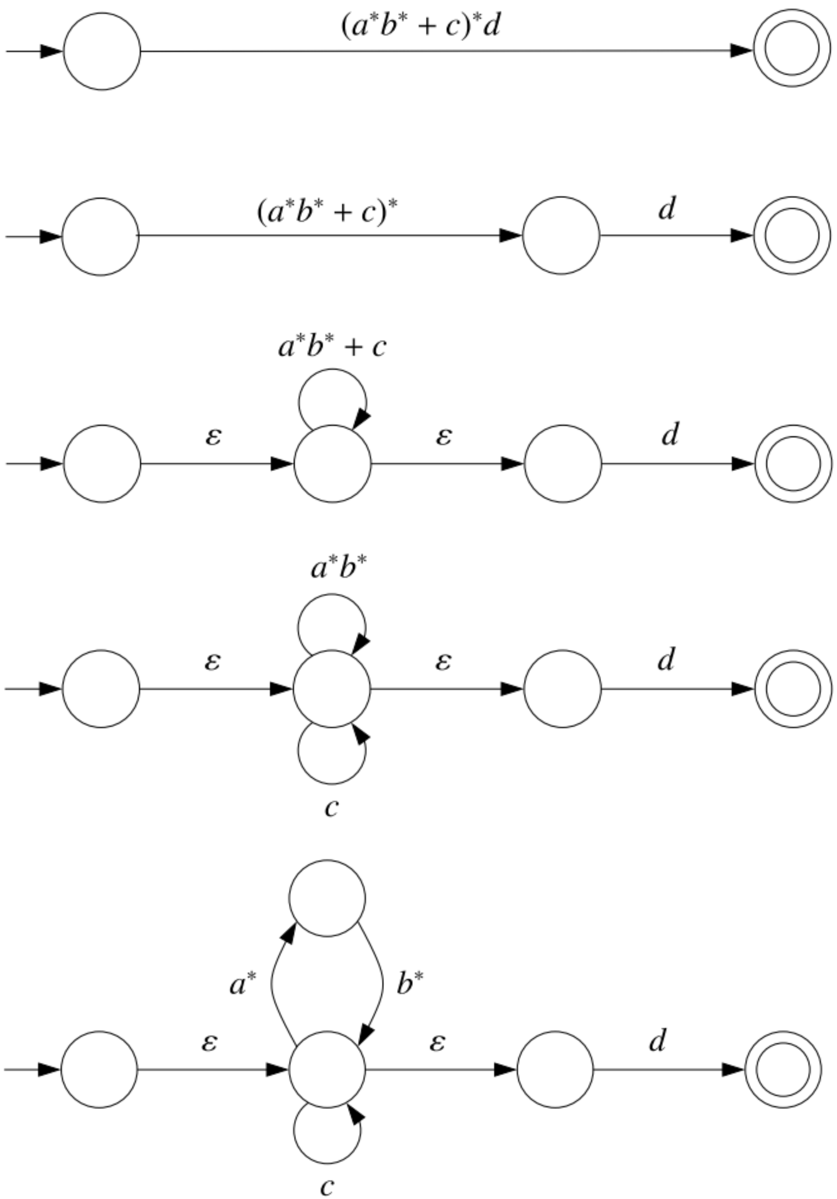


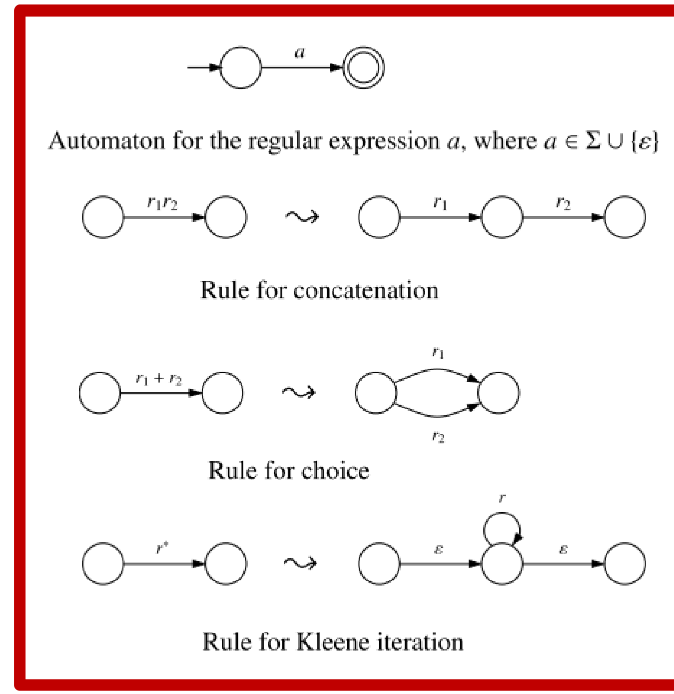
Regular expressions to NFAϵ

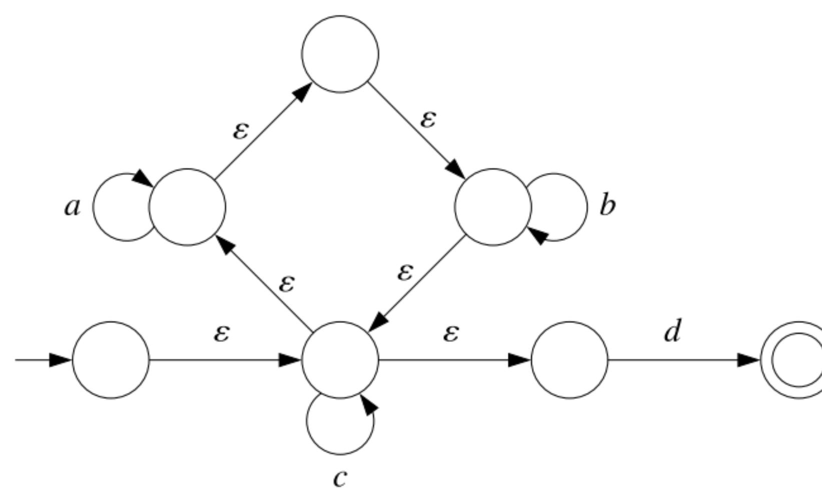




Regular expressions to NFAϵ

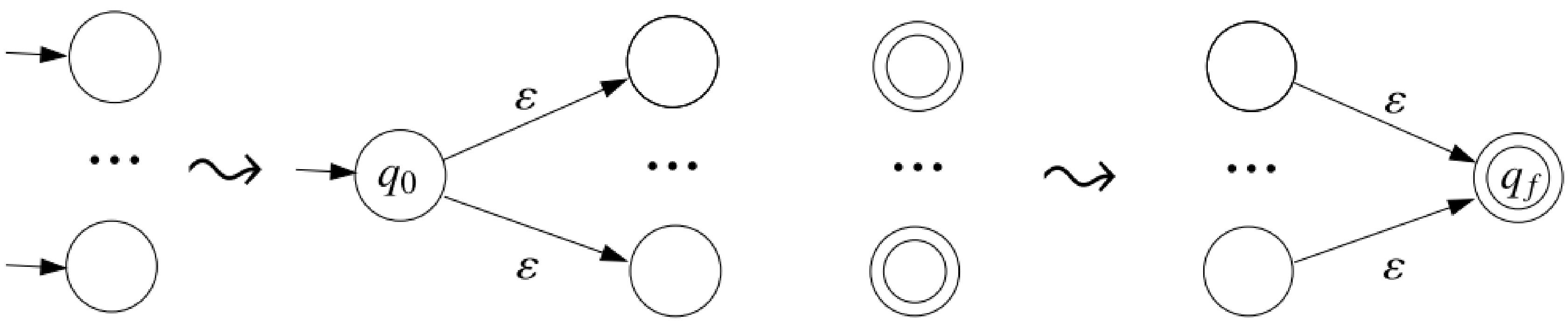






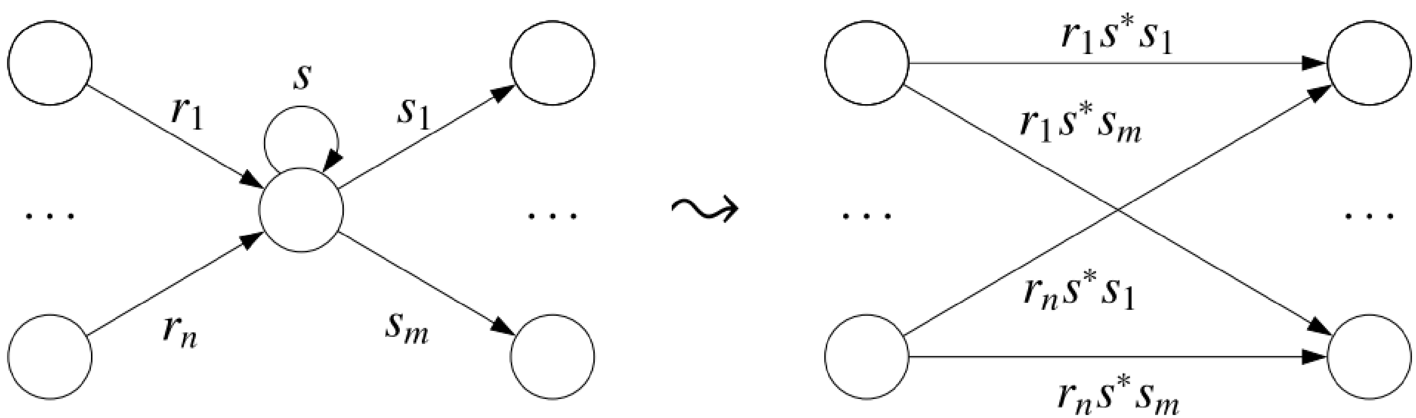
NFAϵ to regular expressions

- Preprocessing: convert into an NFA-ϵ with
  - one initial state without input transitions, and
  - one final state without output transitions.

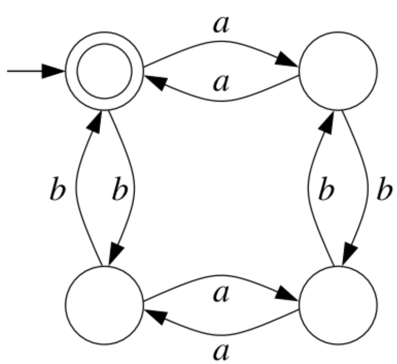


NFAϵ to regular expressions

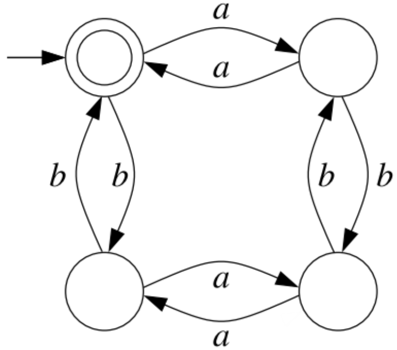
- Processing: apply the following two rules, given priority to the first one.

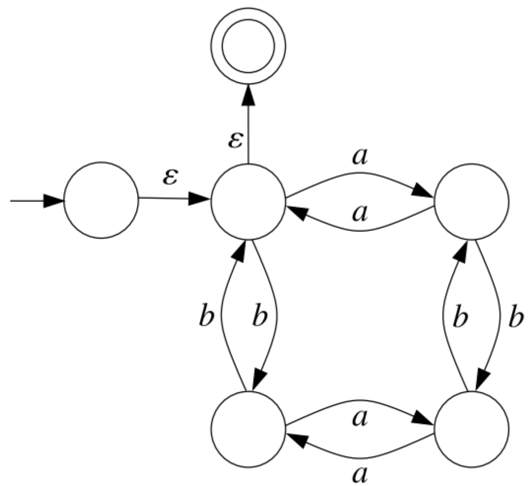


NFAϵ to regular expressions

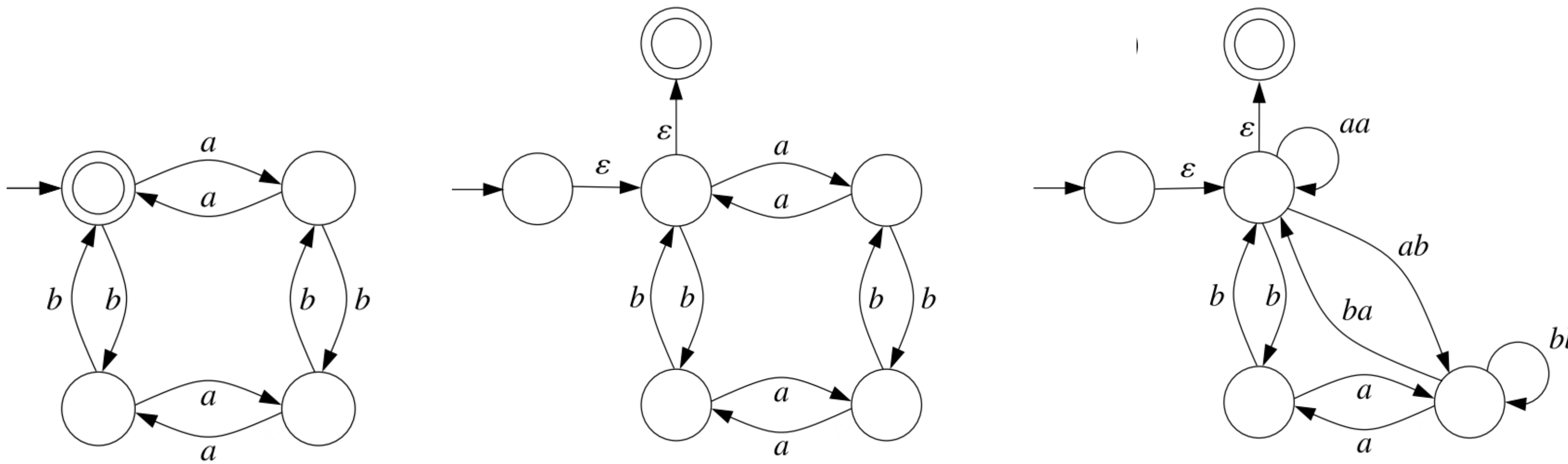


NFAϵ to regular expressions

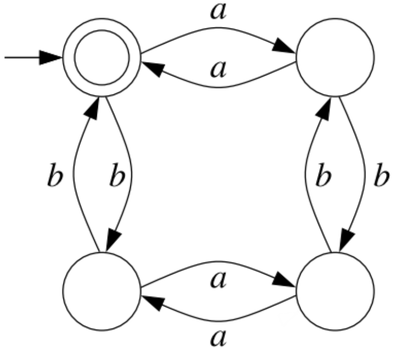


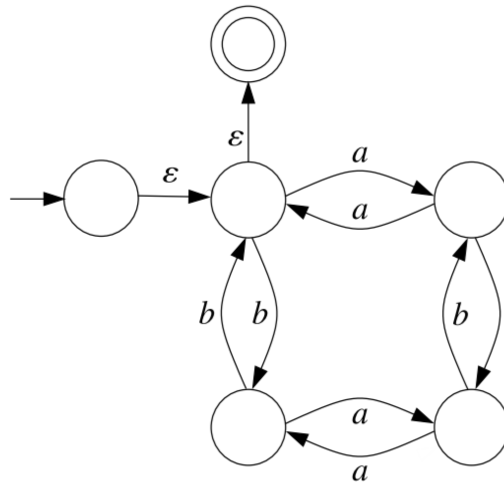


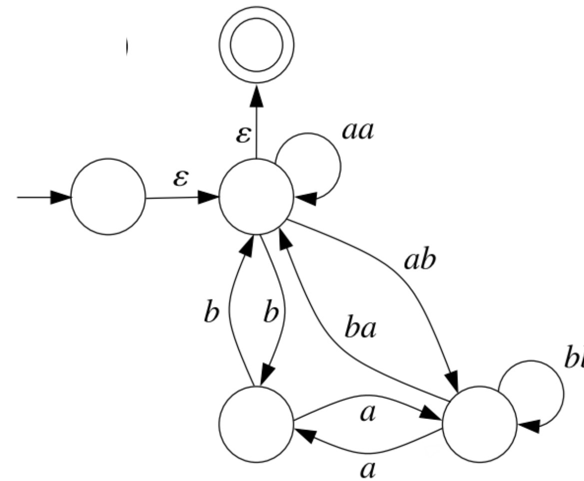
NFAϵ to regular expressions

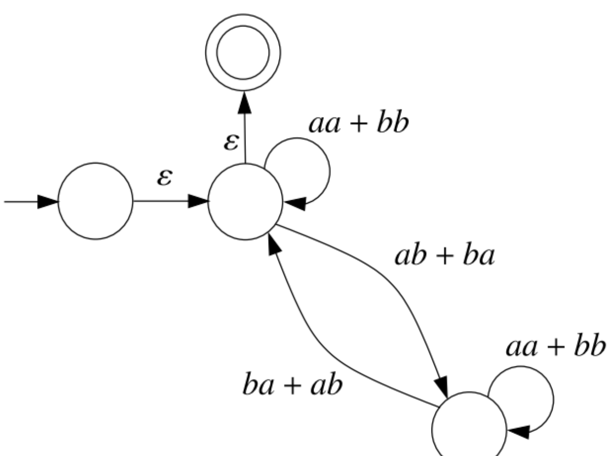


NFAϵ to regular expressions

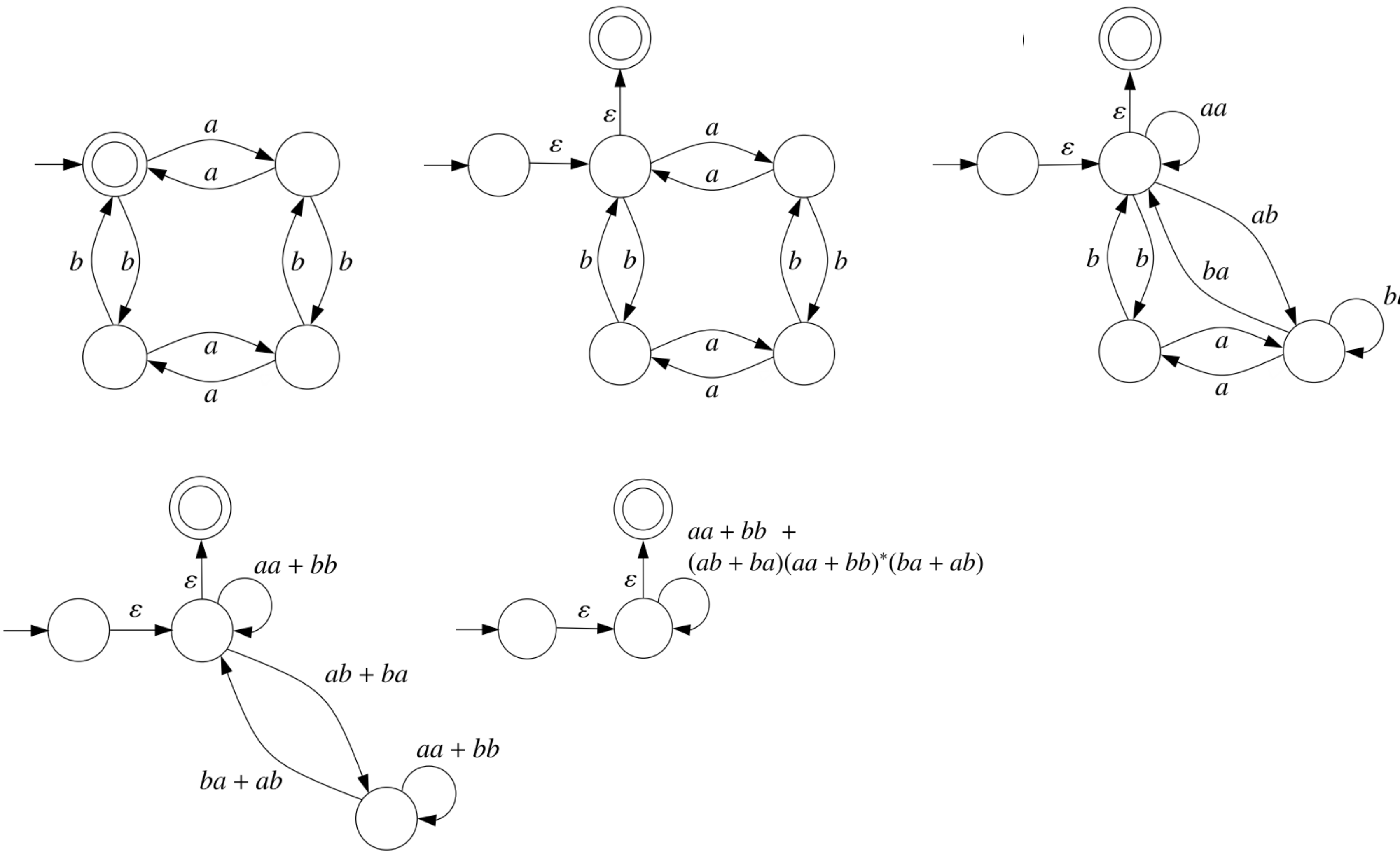




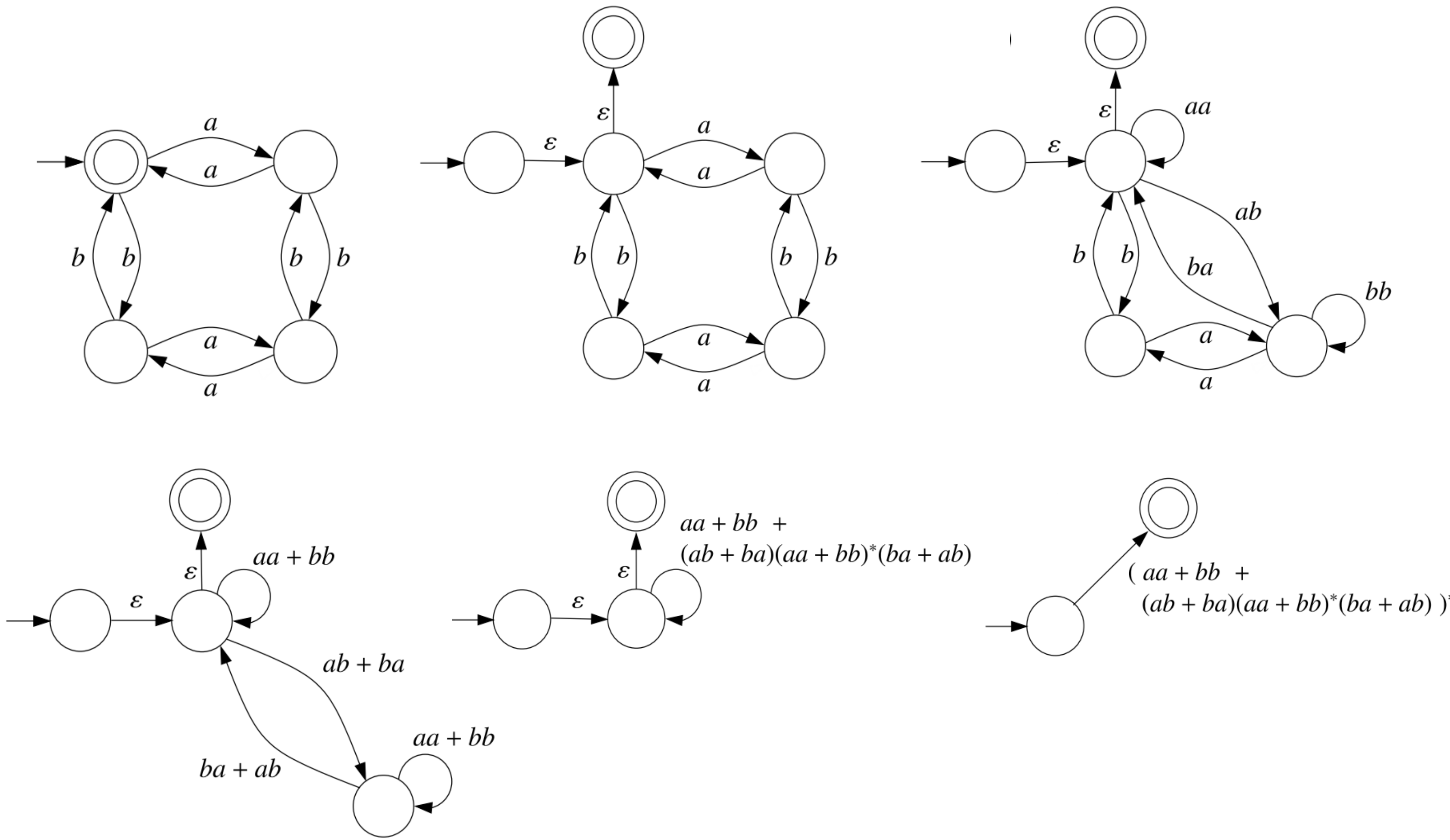




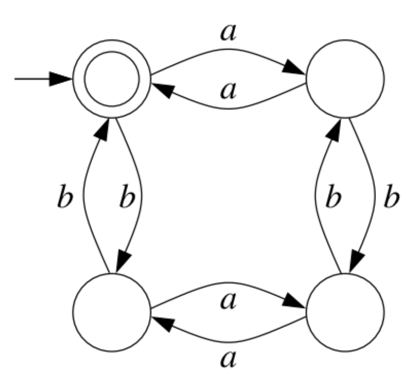
NFAϵ to regular expressions



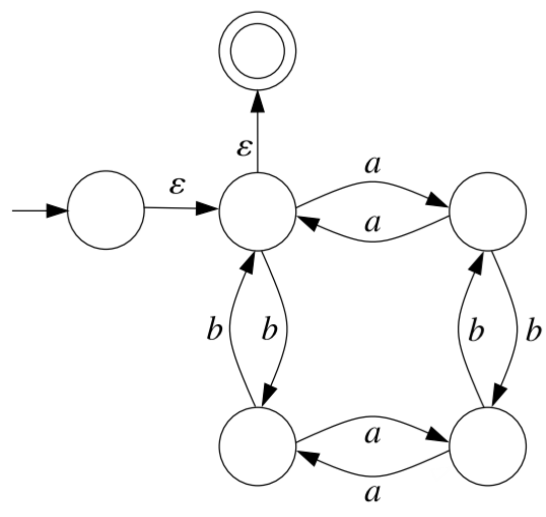
NFAϵ to regular expressions



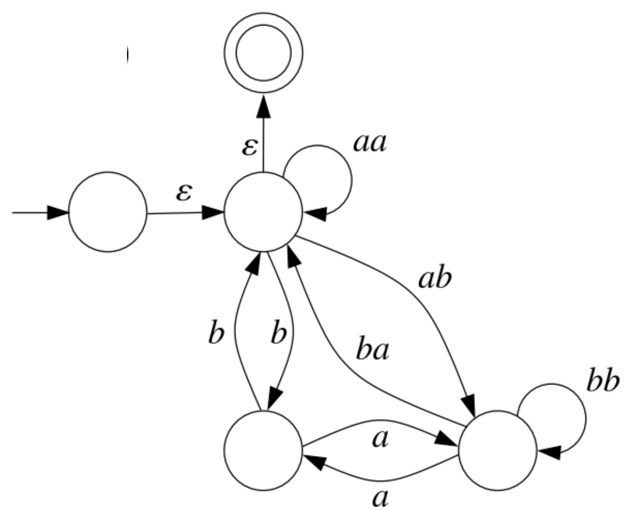
A Tour of Conversions



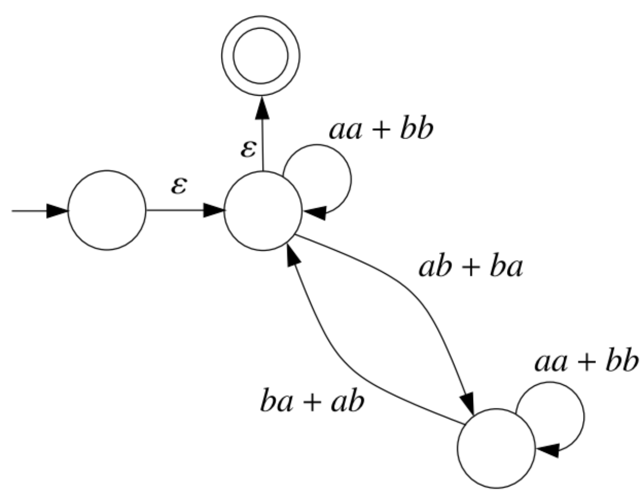
A Tour of Conversions



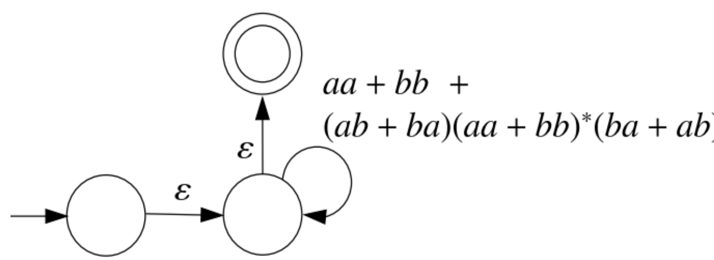
A Tour of Conversions



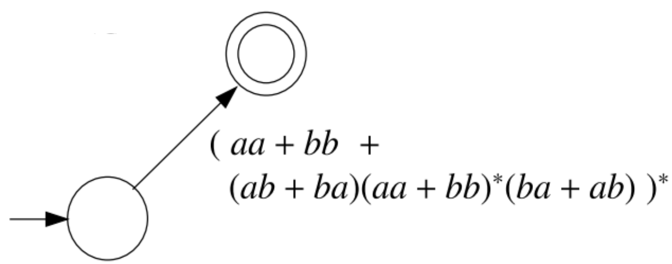
A Tour of Conversions



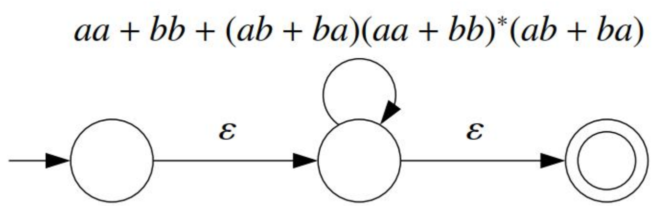
A Tour of Conversions



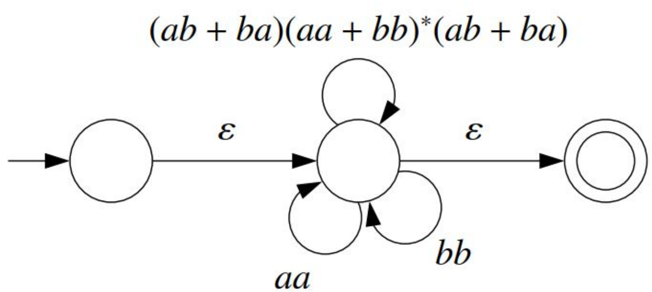
A Tour of Conversions



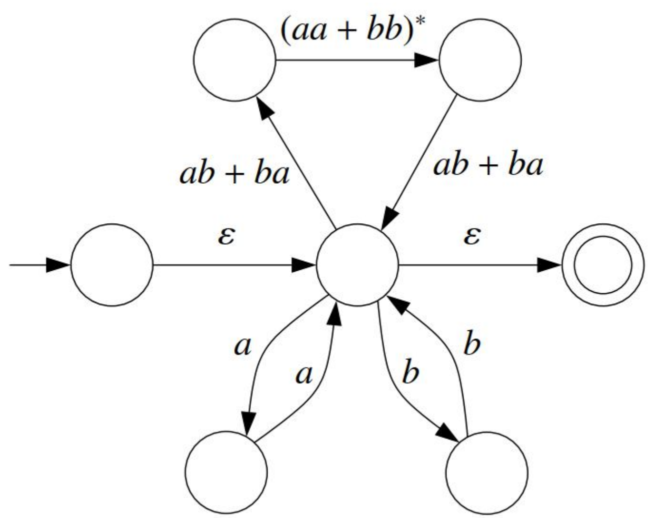
A Tour of Conversions



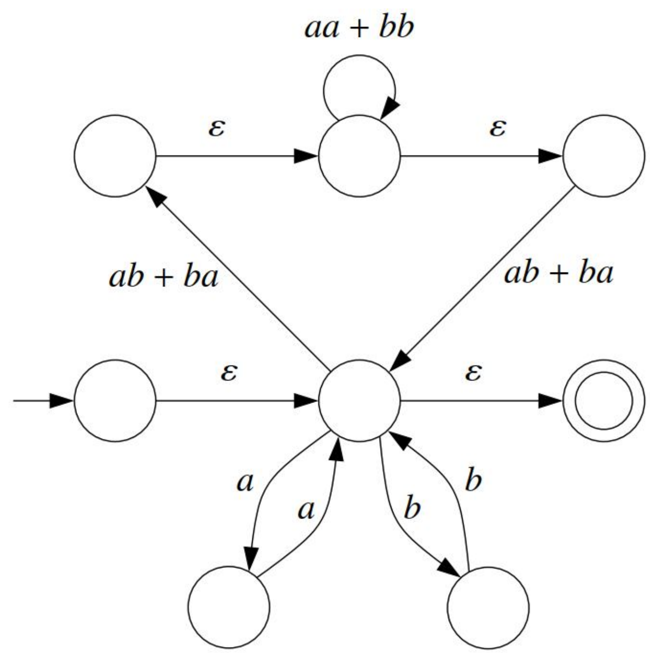
A Tour of Conversions



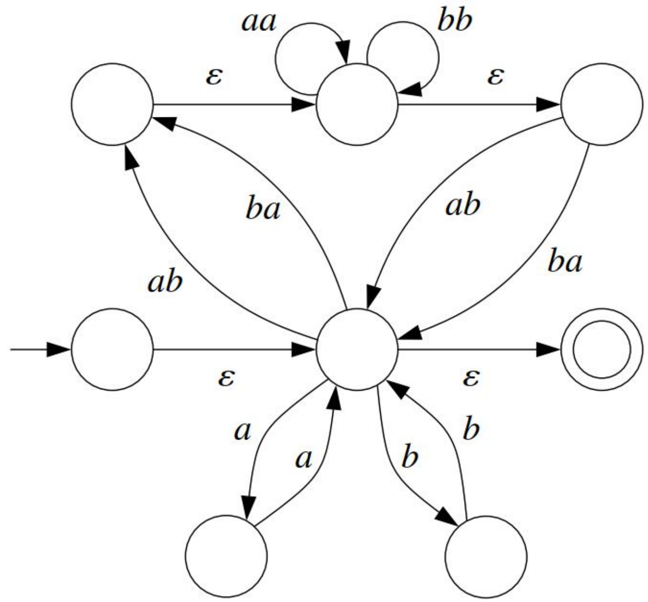
A Tour of Conversions



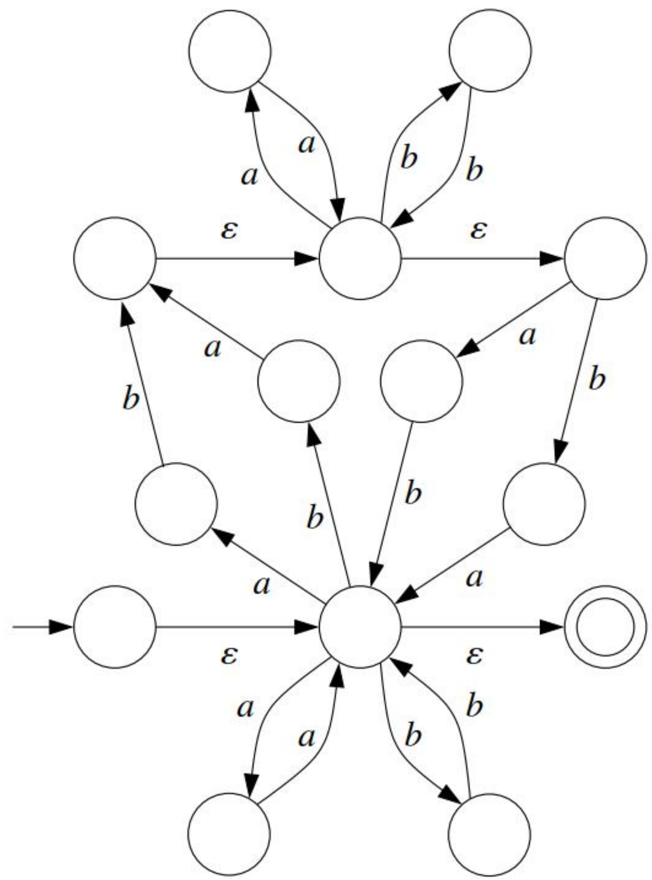
A Tour of Conversions



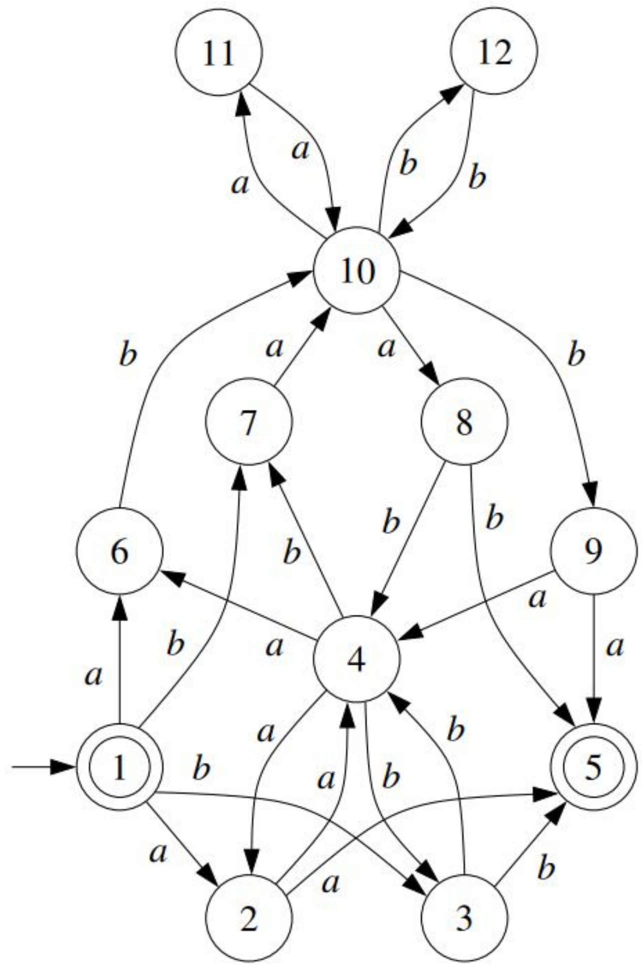
A Tour of Conversions



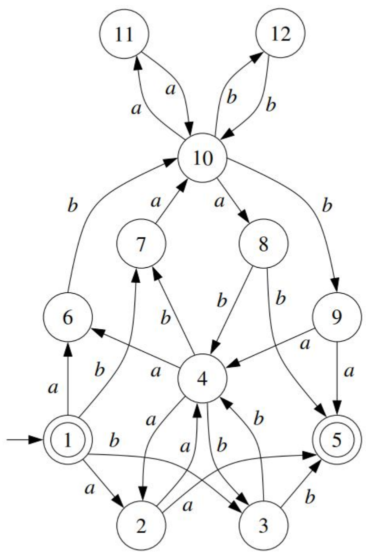
A Tour of Conversions



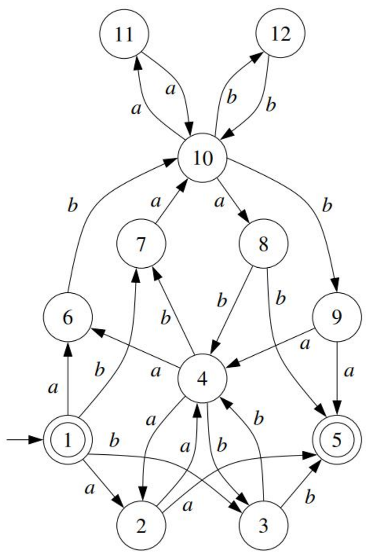
A Tour of Conversions

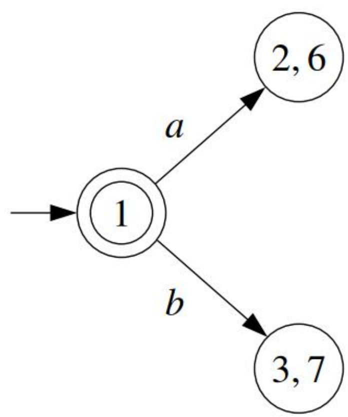


A Tour of Conversions

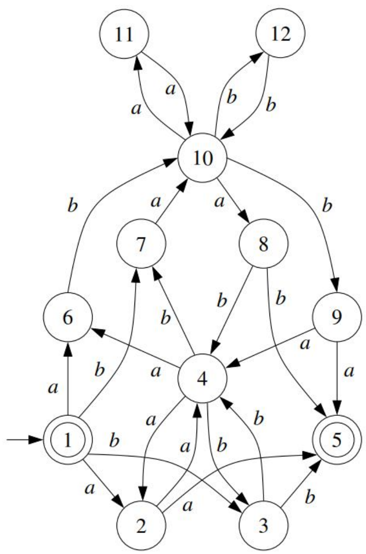


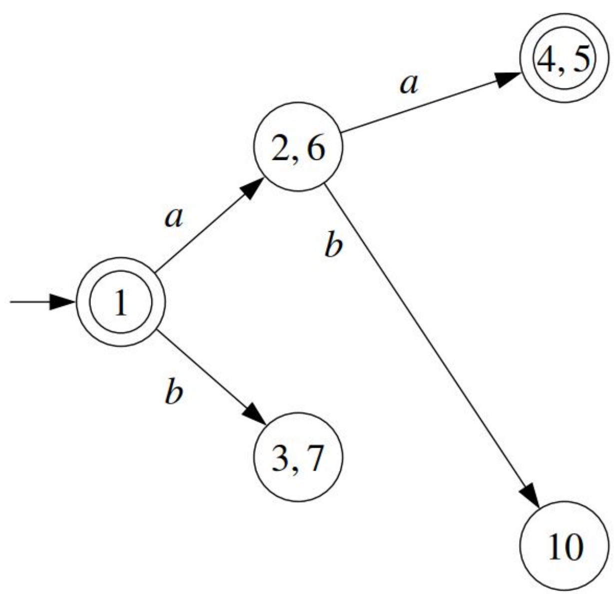
A Tour of Conversions



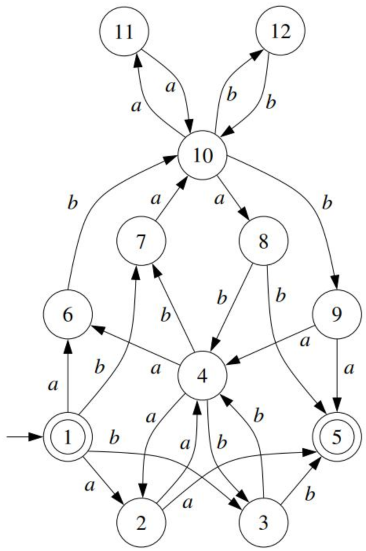


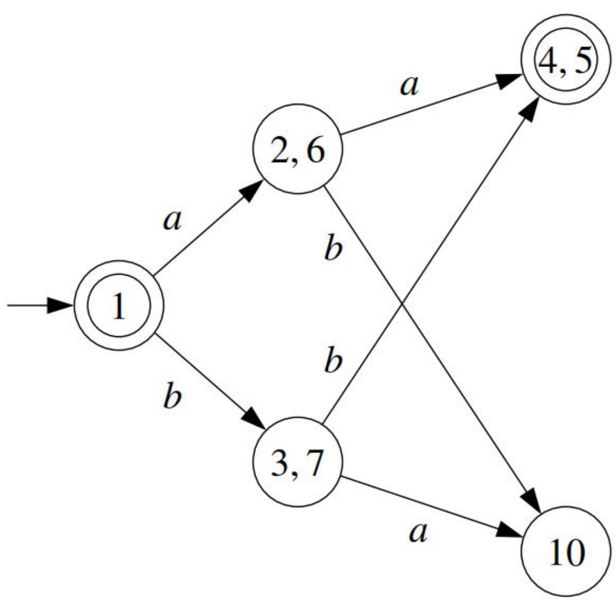
A Tour of Conversions



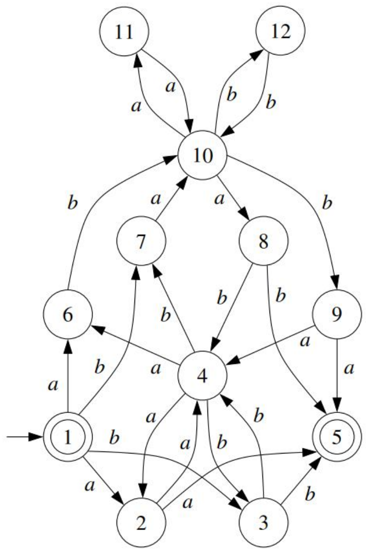


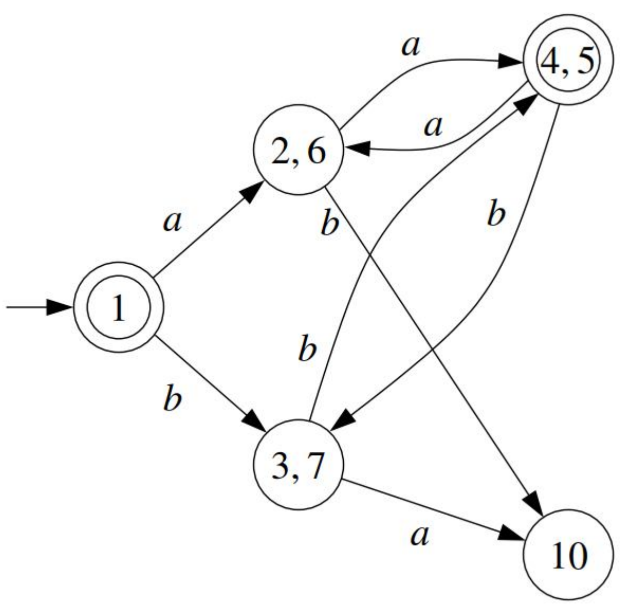
A Tour of Conversions



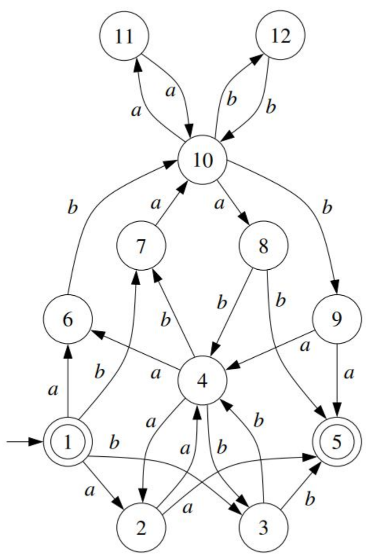


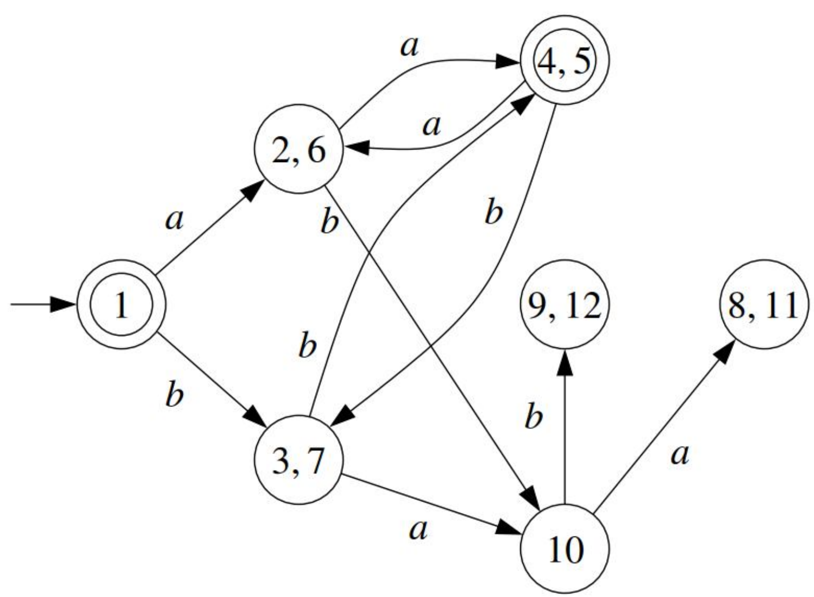
A Tour of Conversions



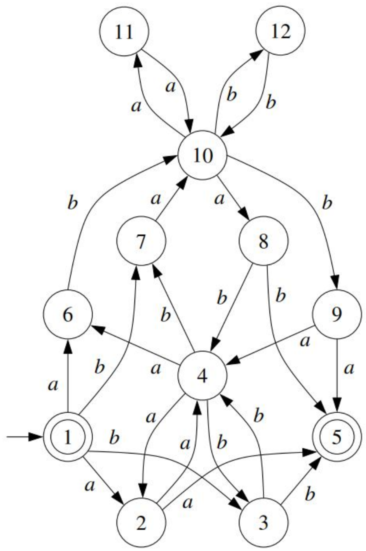


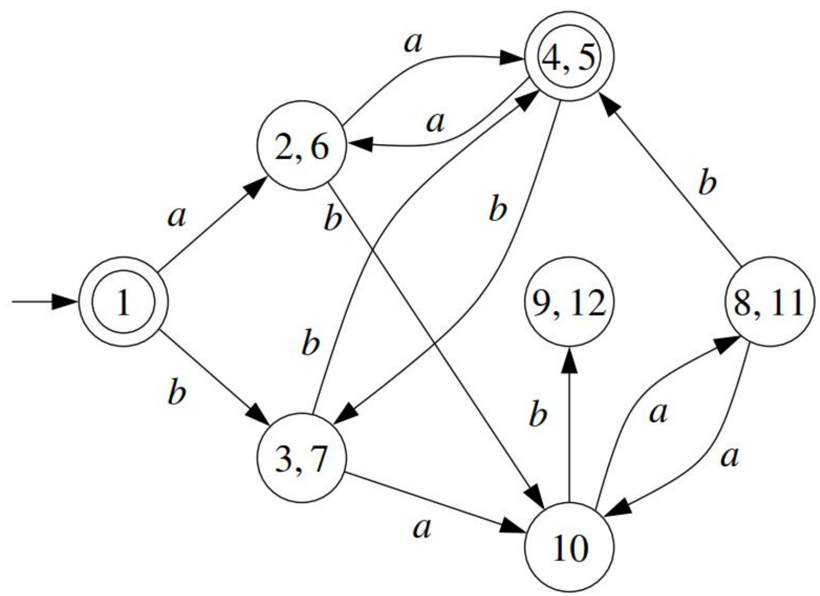
A Tour of Conversions





A Tour of Conversions





A Tour of Conversions