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 \( L^{i+1} = L^i L \)
A deterministic finite automaton is a tuple \( A = (Q, \Sigma, \delta, q_0, F) \) where

- \( Q \) is a finite, nonempty set of states
- \( \Sigma \) 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 \( \Sigma \) 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 \( L(A) \).
A nondeterministic automaton is a tuple $A = (Q, \Sigma, \delta, Q_0, F)$ where

- $Q, \Sigma, F$ are as for DFAs
- $\delta: Q \times \Sigma \rightarrow 2^Q$ 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 $a_1 a_2 \ldots a_n \in \Sigma^*$ is a sequence $q_0 q_1 \ldots q_n$ of states such that $q_0 \in Q_0$ and

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

• 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 $\varepsilon$-transitions (NFA$\varepsilon$)

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

- $Q, \Sigma, Q_0, F$ are as for NFAs
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$ is the transition function
 Runs of an NFA$\epsilon$ on a word

- A run of an NFA$\epsilon$ on a word $a_1 a_2 \ldots a_n \in \Sigma^*$ is a sequence $q_0 \ldots q'_0 q_1 \ldots q'_1 q_2 \ldots q'_{n-1} q_n \ldots q'_n$ of states such that $q_0 \in Q_0$ and

$$q_0 \rightarrow \ldots \rightarrow q'_0 \rightarrow q_1 \rightarrow \ldots \rightarrow q'_1 \rightarrow q_2 \ldots q'_{n-1} \rightarrow q_n \rightarrow \ldots \rightarrow q'_n$$

- An NFA$\epsilon$ can have 0, 1, or more runs on the same word, even infinitely many.

- An NFA$\epsilon$ 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

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

![Nondeterministic finite automaton with regular expressions](image)
An NFAreg accepts a word $w$ if there are states $q_0, \ldots, q_n$ and regular expressions $r_1, \ldots, r_n$ 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 \( B = (Q, \Sigma, \Delta, q_0, \mathcal{F}) \) with \( L(B) = L(A) \)

1. \( Q, \Delta, \mathcal{F} \leftarrow \emptyset; q_0 \leftarrow Q_0 \)
2. \( \mathcal{W} = \{Q_0\} \)
3. **while** \( \mathcal{W} \neq \emptyset \) **do**
4.  \hspace{1em} **pick** \( Q' \) **from** \( \mathcal{W} \)
5.  \hspace{1em} **add** \( Q' \) **to** \( Q \)
6.  \hspace{1em} **if** \( Q' \cap F \neq \emptyset \) **then** **add** \( Q' \) **to** \( \mathcal{F} \)
7.  \hspace{1em} **for all** \( a \in \Sigma \) **do**
8.  \hspace{2em} \( Q'' \leftarrow \bigcup_{q \in Q'} \delta(q, a) \)
9.  \hspace{2em} **if** \( Q'' \notin Q \) **then** **add** \( Q'' \) **to** \( \mathcal{W} \)
10. \hspace{2em} **add** \( (Q', a, Q'') \) **to** \( \Delta \)
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 the contrary.

Then two different words of length $n$ lead to the same state. Let the words be $uav_1$ and $uav_2$.

Then $uav_1$ and $uav_2$ lead to the same state too, but only one of the is accepted. Contradiction.
NFA$\varepsilon$ to NFA
NFA$\varepsilon$ to NFA

Saturation
NFA$\varepsilon$ to NFA

Saturation

Check of the initial state + $\varepsilon$-removal
A one-pass algorithm

$NFA_{\varepsilon}toNFA(A)$

**Input:** NFA-\(\varepsilon\) \(A = (Q, \Sigma, \delta, Q_0, F)\)

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

1. \(Q'_0 \leftarrow Q_0\)
2. \(Q' \leftarrow Q_0; \delta' \leftarrow \emptyset; F' \leftarrow F \cap Q_0\)
3. \(\delta'' \leftarrow \emptyset; W \leftarrow \{(q, \alpha, q') \in \delta \mid q \in Q_0\}\)
4. **while** \(W \neq \emptyset\) **do**
   5. **pick** \((q_1, \alpha, q_2)\) from \(W\)
   6. **if** \(\alpha \neq \varepsilon\) **then**
      7. add \(q_2\) to \(Q'\); add \((q_1, \alpha, q_2)\) to \(\delta'\); if \(q_2 \in F\) then add \(q_2\) to \(F'\)
      8. **for all** \(q_3 \in \delta(q_2, \varepsilon)\) **do**
         9. **if** \((q_1, \alpha, q_3) \notin \delta'\) **then** add \((q_1, \alpha, q_3)\) to \(W\)
      10. **for all** \(a \in \Sigma, q_3 \in \delta(q_2, a)\) **do**
          11. **if** \((q_2, a, q_3) \notin \delta'\) **then** add \((q_2, a, q_3)\) to \(W\)
      12. **else** /* \(\alpha = \varepsilon\) */
   13. add \((q_1, \alpha, q_2)\) to \(\delta''\); if \(q_2 \in F\) then add \(q_1\) to \(F'\)
   14. **for all** \(\beta \in \Sigma \cup \{\varepsilon\}, q_3 \in \delta(q_2, \beta)\) **do**
       15. **if** \((q_1, \beta, q_3) \notin \delta' \cup \delta''\) **then** add \((q_1, \beta, q_3)\) to \(W\)
Correctness

**Proposition.** Let $A$ be an NFA$\epsilon$ and let $B = \text{NFA}_\epsilon\text{toNFA}(A)$. 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 $\epsilon$-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 $\epsilon$-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 NFA$\varepsilon$

$$(a^*b^* + c)^*d$$
Regular expressions to NFA$\varepsilon$

• **Preprocessing**: Convert the regular expression into another one which is either equal to $\emptyset$, or does not contain any occurrence of $\emptyset$.

• Use the following rewrite rules:

\[
\begin{align*}
\emptyset \cdot r & \leadsto \emptyset \\
r \cdot \emptyset & \leadsto \emptyset \\
r + \emptyset & \leadsto r \\
\emptyset^* & \leadsto \varepsilon \\
\emptyset + r & \leadsto r
\end{align*}
\]
Regular expressions to NFAε

Automaton for the regular expression \(a\), where \(a ∈ \Sigma \cup \{ε\}\):

- Rule for concatenation
- Rule for choice
- Rule for Kleene iteration

\[(a^*b^* + c)^*d\]
Regular expressions to NFAε

Automaton for the regular expression $a$, where $a \in \Sigma \cup \{\varepsilon\}$

Rule for concatenation

Rule for choice

Rule for Kleene iteration
Regular expressions to NFAε

\[(a^*b^* + c)^*d\]

\[(a^*b^* + c)^*\]

\[a^*b^* + c\]

\[\varepsilon \rightarrow \varepsilon \rightarrow d\]

 Automaton for the regular expression \(a\), where \(a \in \Sigma \cup \{\varepsilon\}\)

Rule for concatenation

Rule for choice

Rule for Kleene iteration
Regular expressions to NFAε

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

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

d

a^*b^* + c

e

e

d

ε

c

Automaton for the regular expression $a$, where $a \in \Sigma \cup \{\varepsilon\}$

Rule for concatenation

Rule for choice

Rule for Kleene iteration
Regular expressions to NFAε

The diagram shows the conversion of regular expressions to NFA (Nondeterministic Finite Automata) with ε-transitions. Each transition represents a component of the regular expression, with ε-transitions allowing for empty transitions.

The automaton for the regular expression $a^*b^* + c^*d$ is illustrated, along with rules for concatenation, choice, and Kleene iteration.

For example, the rule for concatenation is shown with states $r_1$ and $r_2$ connected by transitions $r_1 \rightarrow r_2$. The rule for choice is depicted with states $r_1 + r_2$ connected by transitions $r_1$ and $r_2$. The rule for Kleene iteration is shown with a loop allowing for zero or more repetitions.

Automaton for the regular expression $a$, where $a \in \Sigma \cup \{\varepsilon\}$

Rule for concatenation

Rule for choice

Rule for Kleene iteration
Regular expressions to NFAε

\[(a^*b^* + c)^*d\]

\[(a^*b^* + c)^*\]

\[a^*b^* + c\]

\[\epsilon\]

\[\epsilon\]

\[\epsilon\]

\[\epsilon\]

\[a^*\]

\[b^*\]

\[c\]

\[\epsilon\]

\[\epsilon\]

\[\epsilon\]

\[\epsilon\]

\[a\]

\[b\]

\[c\]

\[d\]
NFA$\varepsilon$ to regular expressions

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

Diagram:

Input states with input transitions to $q_0$, followed by $\varepsilon$ transitions to more states, and finally $\varepsilon$ transitions to the final state $q_f$. The diagram shows the transformation process.
NFA$\varepsilon$ to regular expressions

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

\[
\begin{align*}
r_1 &
\sim
r_1 + r_2 \\
\end{align*}
\]
NFA$\epsilon$ to regular expressions
NFA$\varepsilon$ to regular expressions
NFA$\epsilon$ to regular expressions
NFA$\varepsilon$ to regular expressions
NFAε to regular expressions
NFA_ε to regular expressions

\[
\begin{align*}
(a + b)(a + b)^* & \quad \varepsilon + (ab + ba) + (aa + bb) + (ab + ba)(aa + bb)^*(ba + ab) \\
\varepsilon + (ab + ba) + (aa + bb) + (ab + ba)(aa + bb)^*(ba + ab) & \quad (aa + bb + (ab + ba)(aa + bb)^*(ba + ab))^*
\end{align*}
\]
A Tour of Conversions
A Tour of Conversions
A Tour of Conversions
A Tour of Conversions
A Tour of Conversions

\[
\begin{align*}
&aa + bb + (ab + ba) (aa + bb)^* (ba + ab) \\
&\varepsilon \\
&\varepsilon
\end{align*}
\]
A Tour of Conversions

\[(aa + bb) + (ab + ba)(aa + bb)^*(ba + ab)^*\]
A Tour of Conversions

\[ aa + bb + (ab + ba)(aa + bb)^*(ab + ba) \]
A Tour of Conversions

\[(ab + ba)(aa + bb)^*(ab + ba)\]
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