

Models of Computation

5: Regular expressions, finite automaton

Regular expressions

Applications

- search and replace dialogs of text editors
- search engines
- text processing utilities (e.g. sed and AWK)
- programming languages, lexical analysis
- genom analysis (genom as string)
- spam/malware filter
- ...

Regular expressions

Let V and $V' = \{\varepsilon, \cdot, +, *, (,)\}$ be disjoint alphabets. A **regular expression** over V is defined recursively as follows:

1. ε is a regular expression over V ,
2. all $a \in V$ are regular expressions over V ,
3. If R is a regular expression over V , then R^* is also a regular expression over V ,
4. If Q and R are regular expressions over V , then $(Q \cdot R)$ and $(Q + R)$ are also regular expressions over V .

* denotes the closure of iteration,
· concatenation, and
+ union.

Regular expressions

Each regular expression **represents a regular language**, which is defined as:

1. ϵ represents the language $\{\epsilon\}$,
2. Letter a ($\in V$) represents the language $\{a\}$,
3. if R is a regular expression over V , which represents the language L , then R^* represents L^* ,
4. if R and R' are regular expressions over V , s.t.
 R represents the language L and
 R' represents the language L' , then
 $(R \cdot R')$ represents the language LL' ,
 $(R + R')$ represents the language $L \cup L'$.

Regular expressions

Parentheses can be omitted when defining precedence on operations. The the usual sequence is: $*$, \cdot , $+$. The following regular expressions are equivalent:

- a^* is the same as $(a)^*$ and represent the language $\{a\}^*$.
- $(a + b)^*$ is the same as $((a) + (b))^*$ and represents the language $\{a, b\}^*$.
- $a^* \cdot b$ is the same as $((a)^*) \cdot (b)$ and represents the language $\{a\}^*b$.
- $b + ab^*$ is the same as $(b) + ((a) \cdot (b)^*)$ and represents the language $\{b\} \cup \{a\}\{b\}^*$.
- $(a + b) \cdot a^*$ is the same as $((a) + (b)) \cdot ((a)^*)$ and represents the language $\{a, b\}\{a\}^*$.

Regular expressions

Let P , Q , and R be regular expressions. Then following hold:

- $P + (Q + R) = (P + Q) + R$
- $P \cdot (Q \cdot R) = (P \cdot Q) \cdot R$
- $P + Q = Q + P$
- $P \cdot (Q + R) = P \cdot Q + P \cdot R$
- $(P + Q) \cdot R = P \cdot R + Q \cdot R$
- $P^* = \varepsilon + P \cdot P^*$
- $\varepsilon \cdot P = P \cdot \varepsilon = P$
- $P^* = (\varepsilon + P)^*$

Regular expressions

Example:

The language represented the regular expressions

$(a + b)a^*$ and $aa^* + ba^*$ is the same:

$\{ aa^n \mid n \geq 0 \} \cup \{ ba^n \mid n \geq 0 \}$.

The language represented by $a + ba^*$ is:

$\{ a, b, ba, ba^2, ba^3, \dots \}$.

Expressive power of regular expressions

Theorem:

- 1) Every regular expression represents a regular (type 3) language.
- 2) For every regular (type 3) language, there is a regular expression representing the language.

Proof:

1) follows from the fact that the class of regular languages \mathcal{L}_3 is closed for the regular operations.

Expressive power of regular expressions

Proof:

For 2), we show that for every regular language L generated by a grammar $G = (N, T, P, S)$, a regular expression can be constructed, that represents L .

- Let $N = \{A_1, \dots, A_n\}$, $n \geq 1$, $S = A_1$.
- Assume, each rule of G is of form $A_i \rightarrow aA_j$ or $A_i \rightarrow \varepsilon$, where $a \in T$, $1 \leq i, j \leq n$.
- We say that a non-terminal A_m is **affected** by the derivation $A_i \Rightarrow^* uA_j$ ($u \in T^*$), if A_m occurs in an intermediate string between A_i and uA_j in the derivation.

Expressive power of regular expressions

Proof (cont.):

- A derivation $A_i \Rightarrow^* uA_j$ is called **k -bounded** if $0 \leq m \leq k$ holds for all non-terminals A_m occurring in the derivation.
- Let $E_{i,j}^k = \{u \in T^* \mid \exists A_i \Rightarrow^* uA_j \text{ } k\text{-bounded derivation}\}$.
- We show by induction on k , that for language $E_{i,j}^k$, there is a regular expression representing $E_{i,j}^k$, $0 \leq i,j,k \leq n$.

Expressive power of regular expressions

Proof (cont.):

- $k = 0$ (induction start):
 - For $i \neq j$, $E^0_{i,j}$ is either empty, or it consists of symbols of T ($a \in E^0_{i,j}$ if and only if $A_i \rightarrow aA_j \in P$.)
 - For $i = j$, $E^0_{i,j}$ consists of ε and zero or more elements of T , so $E^0_{i,j}$ can be represented by a regular expression.

Expressive power of regular expressions

Proof (cont.):

- $k-1 \rightarrow k$ (induction step):
 - Assume that for a fixed k , $0 < k \leq n$, $E^{k-1}_{i,j}$ can be represented by a regular expression.
 - Then for all i, j, k it holds that
$$E^k_{i,j} = E^{k-1}_{i,j} + E^{k-1}_{i,k} \cdot (E^{k-1}_{k,k})^* \cdot E^{k-1}_{k,j}.$$
 - Therefore, $E^k_{i,j}$ can also be represented by a regular expression.
 - Let I_ε be the set of indices i for which $A_i \rightarrow \varepsilon$.
 - Then $L(G) = \bigcup_{i \in I_\varepsilon} E^n_{1,i}$ can be represented by a regular expression. The claim of the theorem follows. □

Finite Automata (FA)

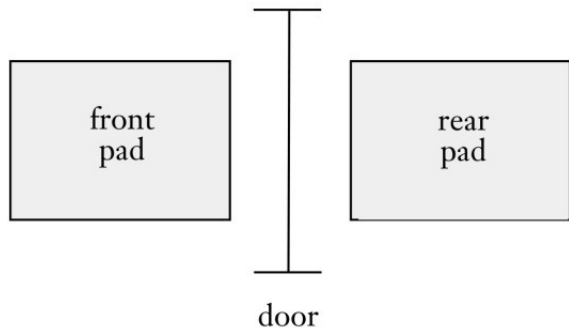
- Identifying formal languages is also possible with recognition devices, i.e. by automata.
- An automaton can process and identify words.
- Grammars use a synthesizing approach, while automata an analytic one.
- An automaton accepts or rejects an input word.

Finite Automata (FA)

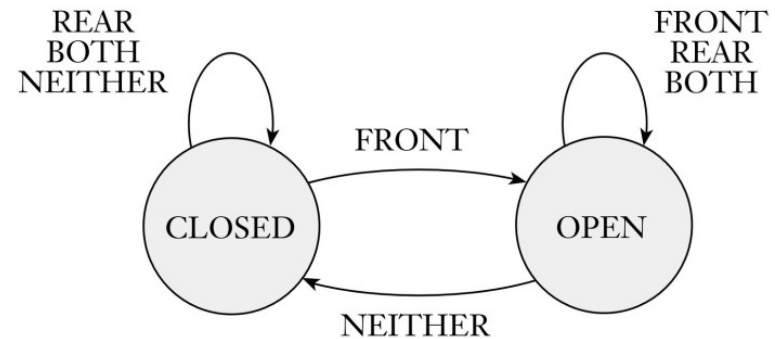
- A finite automaton (FA) performs a sequence of steps in discrete time intervals
- The FA starts in the initial state.
- The input word is located on the input tape and the reading head is on the leftmost symbol of an input word.
- After reading a symbol, the FA moves the reading head to one position to the right, then the state changes, regarding the state transition function.
- If the FA has read the input, it stops, accepts or rejects the input.

Finite Automata (FA)

- Example: automatic door control



State transition diagram:



State transition table:

		input signal			
		NEITHER	FRONT	REAR	BOTH
state	CLOSED	CLOSED	OPEN	CLOSED	CLOSED
	OPEN	CLOSED	OPEN	OPEN	OPEN

Finite Automata (FA)

- Application examples:
 - Automatic door control
 - Coffee machine
 - Pattern recognition
 - Markov chains
 - Pattern recognition
 - Speech processing
 - Optical character recognition
 - Predictions of share prizes in the stock exchange
 - ...

Finite Automata (FA)

A finite automaton is a 5-tuple $A = (Q, T, \delta, q_0, F)$, where

- Q is a finite, nonempty set of **states**,
- T is the finite **alphabet of input symbols**,
- $\delta : Q \times T \rightarrow Q$ is the **state transition function**
- $q_0 \in Q$ is the **initial state** or **start state**,
- $F \subseteq Q$ is the set of **acceptance states** or **end states**.

Finite Automata (FA)

Remark:

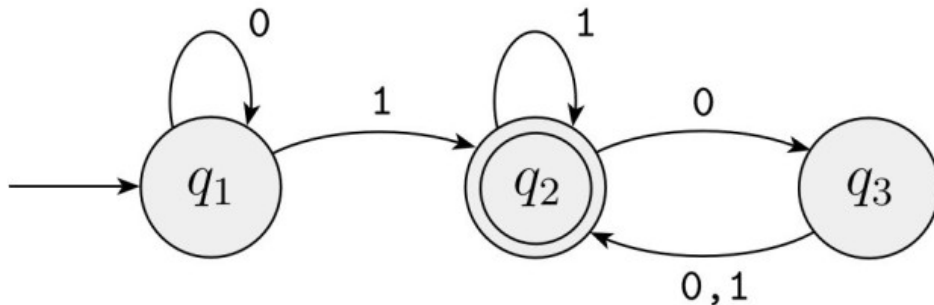
- The function δ can be extended to a function $\hat{\delta} : Q \times T^* \rightarrow Q$ as follows:
 - $\hat{\delta}(q, \varepsilon) = q,$
 - $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$ for all $x \in T^*$ and $a \in T.$

Finite Automata (FA)

Example:

- Let $A = (Q, T, \delta, q_1, F)$ be a FA, where $Q = \{q_1, q_2, q_3\}$, $T = \{0, 1\}$, $F = \{q_2\}$, and $\delta(q_1, 0) = q_1$, $\delta(q_1, 1) = q_2$, $\delta(q_2, 0) = q_3$, $\delta(q_2, 1) = q_2$, $\delta(q_3, 0) = \delta(q_3, 1) = q_2$.

State transition diagram:



State transition table:

δ	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

- The accepted language is $L(A) = \{w \mid w \text{ contains at least one } 1 \text{ and the last } 1 \text{ is not followed by an odd number of } 0\text{s}\}$

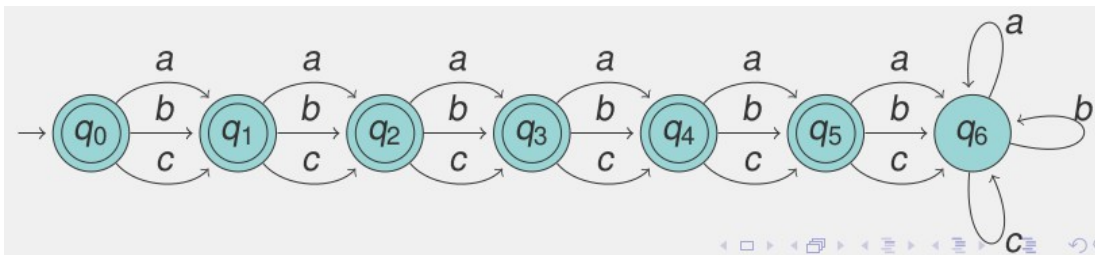
Finite Automata (FA)

Example:

- Let $T = \{a, b, c\}$.
Define a FA, which accepts the words of length of at most 5.

Solution:

- Formaly:
 $A = (\{q_0, \dots, q_6\}, \{a, b, c\}, \delta, q_0, \{q_0, \dots, q_5\})$,
 $\delta(q_i, t) = q_{i+1}$, for $i = 0, \dots, 5, t \in \{a, b, c\}$,
 $\delta(q_6, t) = q_6$, for $t \in \{a, b, c\}$
- State transition diagram:

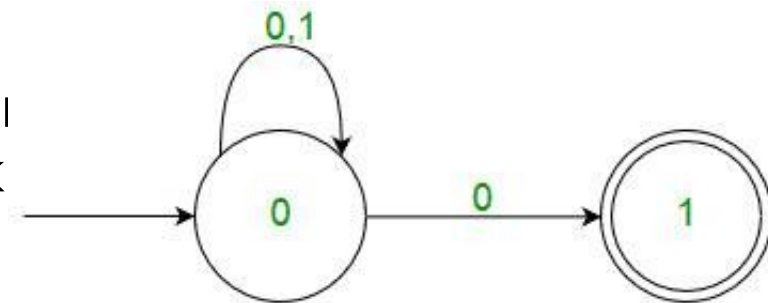


State transition table:

	a	b	c
$\hookrightarrow q_0$	q_1	q_1	q_1
$\leftarrow q_1$	q_2	q_2	q_2
$\leftarrow q_2$	q_3	q_3	q_3
$\leftarrow q_3$	q_4	q_4	q_4
$\leftarrow q_4$	q_5	q_5	q_5
$\leftarrow q_5$	q_6	q_6	q_6
q_6	q_6	q_6	q_6

Deterministic and non-deterministic finite automata

- **Deterministic finite automaton (DFA):**
Function δ is single-valued, i.e. $\forall (q, a) \in Q \times T$ there is exactly one state s , s.t. $\delta(q, a) = s$.
- **Nondeterministic finite automaton (NFA):**
 - Function δ is multi-valued, i.e. $\delta : Q \times T \rightarrow 2^Q$.
 - Multiple initial states are allowed (the set of initial states $Q_0 \subseteq Q$).
 - It is allowed that $\delta(q, a) = \emptyset$ for some (q, a) , i.e. the machine gets stuck
 - Null (or ϵ) move is allowed, i.e. it can move forward without reading symbols.



NFA example

Deterministic and non-deterministic FA

- New features of non-determinism
 - Multiple paths are possible (multiple choices at each step).
 - ϵ -transition is a “free” move without reading input.
 - Accepts the input if some path leads to an accepting state.

Deterministic and non-deterministic FA

- Alternative notation:
- State transitions can also be given in the form $qa \rightarrow p$, where $p \in \delta(q, a)$.
- Let M_δ be set of rules of the state transition of an NFA $A = (Q, T, \delta, Q_0, F)$.
- If M_δ contains exactly one rule $qa \rightarrow p$ for each pair (q, a) , then the FA is deterministic, otherwise non-deterministic.

FA – reduction

- Let $A = (Q, T, \delta, Q_0, F)$ be a FA and $u, v \in QT^*$ words. The FA A **reduces** the u **in one step (directly)** to v (notation: $u \Rightarrow_A v$, or short: $u \Rightarrow v$), if there are a rule $qa \rightarrow p \in M_\delta$ (i.e. $\delta(q, a) = p$) and a word $w \in T^*$, s.t. $u = qaw$ and $v = pw$ hold.
- The FA $A = (Q, T, \delta, Q_0, F)$ **reduces** $u \in QT^*$ to $v \in QT^*$ (notation: $u \Rightarrow_A^* v$, or short: $u \Rightarrow^* v$), if
 - either $u = v$,
 - or \exists a word $z \in QT^*$, s.t. $u \Rightarrow^* z$ and $z \Rightarrow v$.
- Remark: \Rightarrow^* is the reflexive, transitive closure of \Rightarrow .

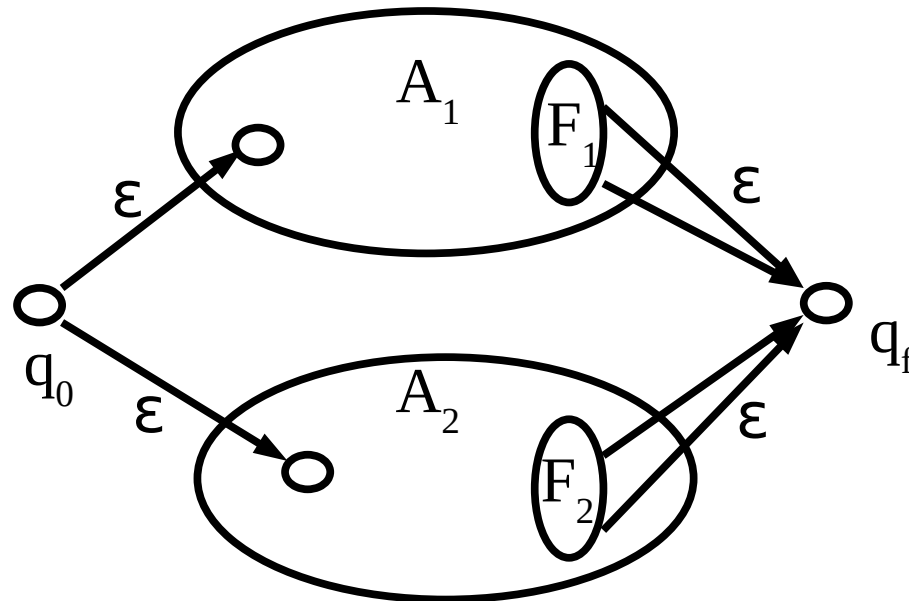
FA – accepted language

- The **language accepted/recognized** by the FA $A = (Q, T, \delta, Q_0, F)$ is:
 $L(A) = \{u \in T^* \mid q_0 u \Rightarrow^* p \text{ for some } q_0 \in Q_0 \text{ and } p \in F\}$
- For a DFA A , there is one single start state $Q_0 = \{q_0\}$. The language accepted by DFA A is:
 $L(A) = \{u \in T^* \mid q_0 u \Rightarrow^* p \text{ for some } p \in F\}$

NFA accepting $L_1 \cup L_2$

Theorem: If L_1 and L_2 are regular languages, then $L_1 \cup L_2$ is also a regular language.

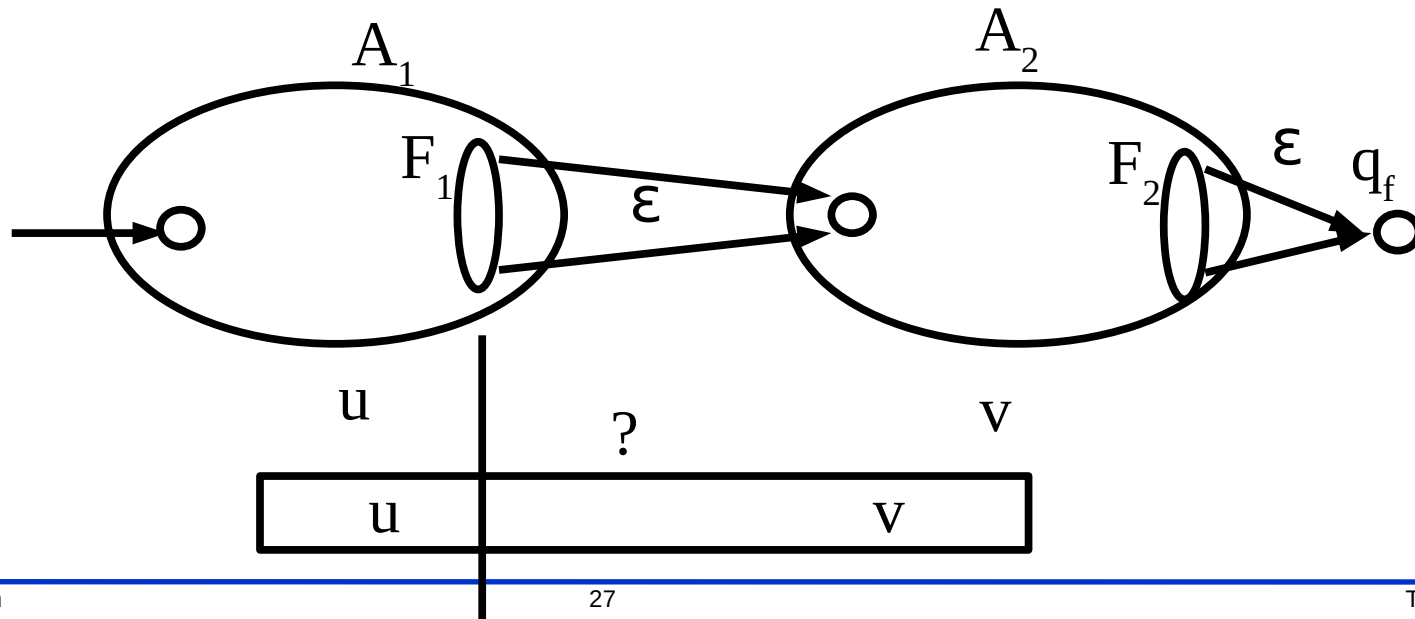
Proof (sketch): Let A_1 be a DFA, accepting L_1 and A_2 a DFA accepting L_2 . Then the following NFA accepts $L_1 \cup L_2$.



NFA accepting L_1L_2

Theorem: If L_1 and L_2 regular languages, then L_1L_2 is also a regular language.

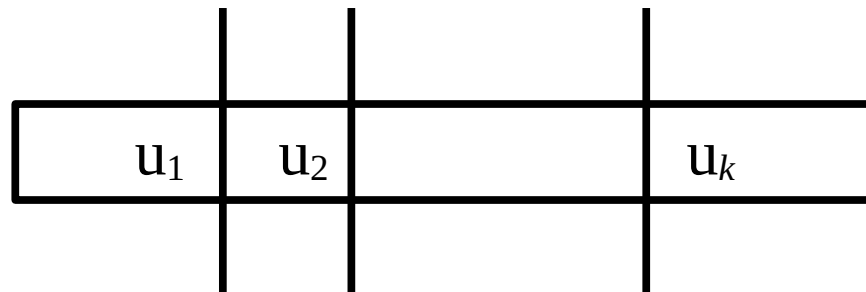
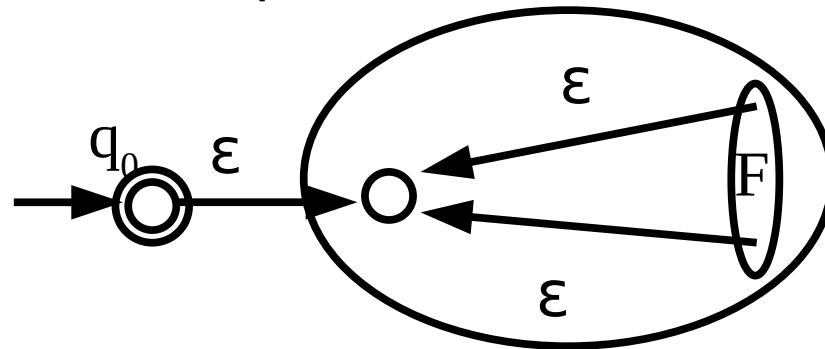
Proof (sketch): Let A_1 be a DFA accepting L_1 , A_2 egy DFA accepting L_2 .
The following NFA accepts L_1L_2 .



NFA accepting L^*

Theorem: If L is a regular language, then L^* is also a regular language.

Proof. (sketch): Let A be a DFA accepting L .
The following NFA accepts L^* .



Computing power of NFA

- **Theorem:** For all NFA $A = (Q, T, \delta, Q_0, F)$ a DFA $A' = (Q', T, \delta', q'_0, F')$ can be constructed, s.t. $L(A) = L(A')$ holds.
- Idea: DFA keeps track of the subset of possible states in NFA
- Remark: In worst case $|Q'| = 2^{|Q|}$.

Computing power of NFA

Proof:

- Let $Q' = 2^Q$ be the set of all subsets of the set Q . (the number of elements of Q' is $2^{|Q|}$).
- Let $\delta' : Q' \times T \rightarrow Q'$ be the function defined as:
 $\delta'(q', a) = \bigcup_{q \in q'} \delta(q, a)$.
- Let $q'_0 = Q_0$ and $F' = \{q' \in Q' \mid q' \cap F \neq \emptyset\}$
- To prove $L(A) \subseteq L(A')$ we prove Lemma 1, to $L(A') \subseteq L(A)$ we prove Lemma 2.
- First, an example (next slide)

NFA – DFA

Example:

- Let $A = (Q, T, \delta, Q_0, F)$ be a NFA, where
 $Q = \{q_0, q_1, q_2\}$, $T = \{a, b\}$, $Q_0 = \{q_0\}$, $F = \{q_2\}$.
 δ is defined as:
 $\delta(q_0, a) = \{q_0, q_1\}$, $\delta(q_0, b) = \{q_1\}$,
 $\delta(q_1, a) = \emptyset$, $\delta(q_1, b) = \{q_2\}$,
 $\delta(q_2, a) = \{q_0, q_1, q_2\}$, $\delta(q_2, b) = \{q_1\}$.
Construct a DFA A' equivalent with A .

Solution:

- DFA: $A' = (Q', T, \delta', q'_0, F')$, where
 $Q' = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$,
 $q'_0 = \{q_0\}$,
 $F' = \{\{q_2\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$,
 δ' next slide

NFA – DFA

Example (cont.):

- δ :
$$\begin{aligned}\delta(q_0, a) &= \{q_0, q_1\}, & \delta(q_0, b) &= \{q_1\}, \\ \delta(q_1, a) &= \emptyset, & \delta(q_1, b) &= \{q_2\}, \\ \delta(q_2, a) &= \{q_0, q_1, q_2\}, & \delta(q_2, b) &= \{q_1\}.\end{aligned}$$

- δ' :
$$\begin{aligned}\delta'(\emptyset, a) &= \emptyset, & \delta'(\emptyset, b) &= \emptyset, \\ \delta'(\{q_0\}, a) &= \{q_0, q_1\}, & \delta'(\{q_0\}, b) &= \{q_1\}, \\ \delta'(\{q_1\}, a) &= \emptyset, & \delta'(\{q_1\}, b) &= \{q_2\}, \\ \delta'(\{q_2\}, a) &= \{q_0, q_1, q_2\}, & \delta'(\{q_2\}, b) &= \{q_1\}, \\ \delta'(\{q_0, q_1\}, a) &= \{q_0, q_1\}, & \delta'(\{q_0, q_1\}, b) &= \{q_1, q_2\}, \\ \delta'(\{q_0, q_2\}, a) &= \{q_0, q_1, q_2\}, & \delta'(\{q_0, q_2\}, b) &= \{q_1\}, \\ \delta'(\{q_1, q_2\}, a) &= \{q_0, q_1, q_2\}, & \delta'(\{q_1, q_2\}, b) &= \{q_1, q_2\}, \\ \delta'(\{q_0, q_1, q_2\}, a) &= \{q_0, q_1, q_2\}, & \delta'(\{q_0, q_1, q_2\}, b) &= \{q_1, q_2\}.\end{aligned}$$

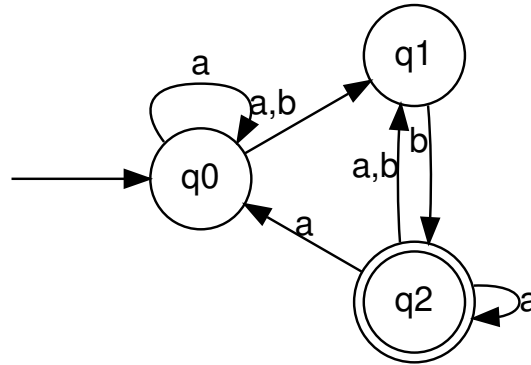
NFA – DFA

Example (cont.):

NVA

$$\begin{aligned} \delta(q_0, a) &= \{q_0, q_1\}, & \delta(q_0, b) &= \{q_1\}, \\ \delta(q_1, a) &= \emptyset, & \delta(q_1, b) &= \{q_2\}, \\ \delta(q_2, a) &= \{q_0, q_1, q_2\}, & \delta(q_2, b) &= \{q_1\}. \end{aligned}$$

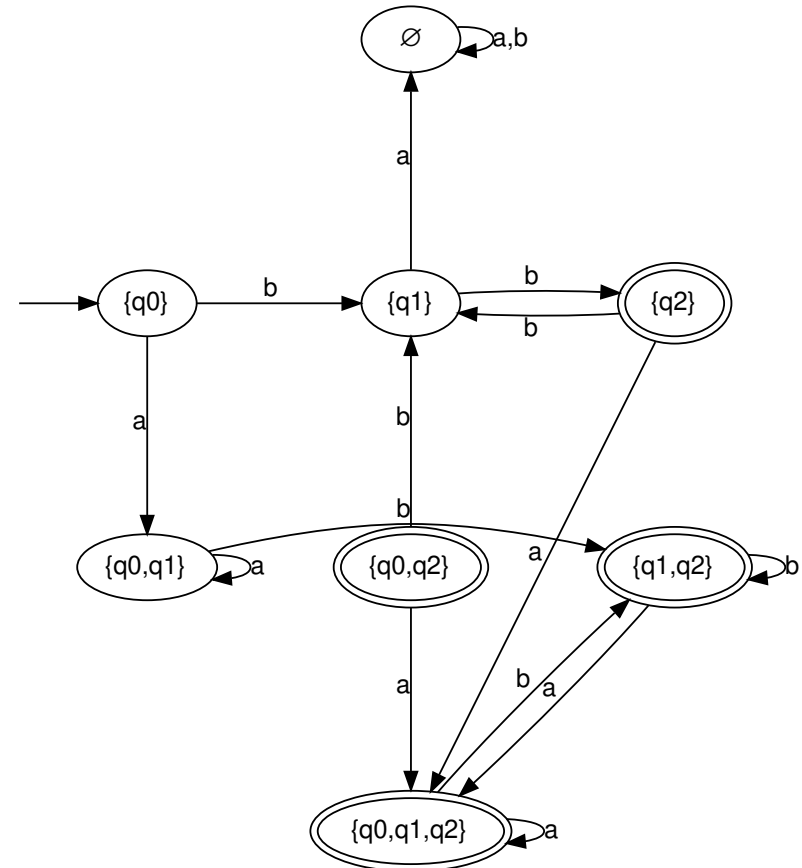
$$F = \{q_2\}$$



DVA

$$\begin{aligned} \delta'(\{\emptyset, a\}) &= \emptyset, & \delta'(\{\emptyset, b\}) &= \emptyset, \\ \delta'(\{\{q_0\}, a\}) &= \{q_0, q_1\}, & \delta'(\{\{q_0\}, b\}) &= \{q_1\}, \\ \delta'(\{\{q_1\}, a\}) &= \emptyset, & \delta'(\{\{q_1\}, b\}) &= \{q_2\}, \\ \delta'(\{\{q_2\}, a\}) &= \{q_0, q_1, q_2\}, & \delta'(\{\{q_2\}, b\}) &= \{q_1\}, \\ \delta'(\{\{q_0, q_1\}, a\}) &= \{q_0, q_1\}, & \delta'(\{\{q_0, q_1\}, b\}) &= \{q_1, q_2\}, \\ \delta'(\{\{q_0, q_2\}, a\}) &= \{q_0, q_1, q_2\}, & \delta'(\{\{q_0, q_2\}, b\}) &= \{q_1\}, \\ \delta'(\{\{q_1, q_2\}, a\}) &= \{q_0, q_1, q_2\}, & \delta'(\{\{q_1, q_2\}, b\}) &= \{q_1, q_2\}, \\ \delta'(\{\{q_0, q_1, q_2\}, a\}) &= \{q_0, q_1, q_2\}, & \delta'(\{\{q_0, q_1, q_2\}, b\}) &= \{q_1, q_2\}. \end{aligned}$$

$$F' = \{\{q_2\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$$



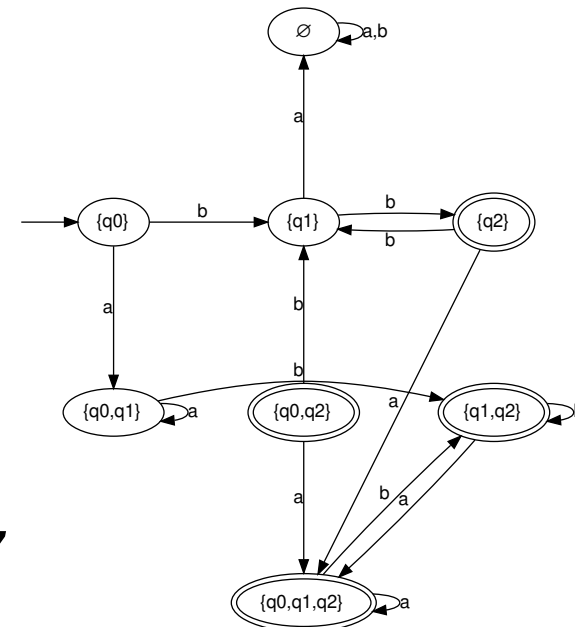
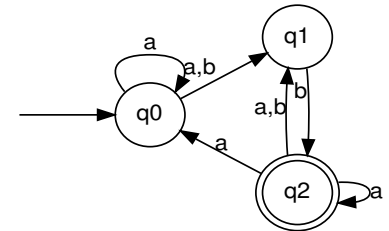
Computing power of NFA

Lemma 1:

- For all $p, q \in Q$, $q' \in Q'$ és $u, v \in T^*$, if $qu \Rightarrow^*_A pv$ and $q \in q'$, then $\exists p' \in Q'$, s.t. $q'u \Rightarrow^*_{A'} p'v$ and $p \in p'$.

Proof:

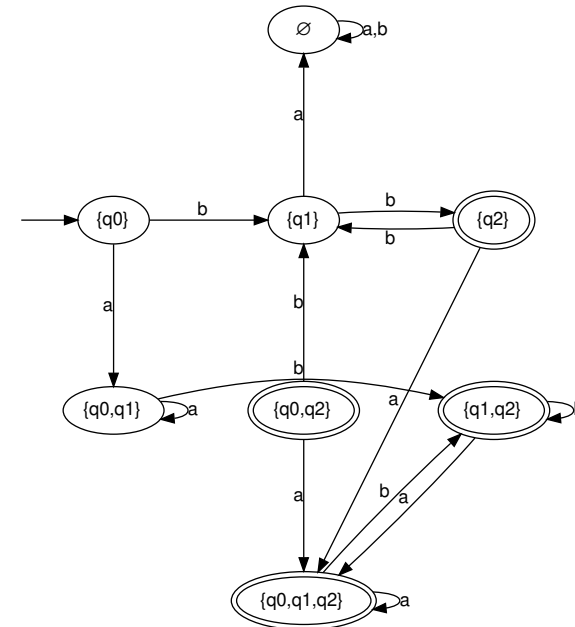
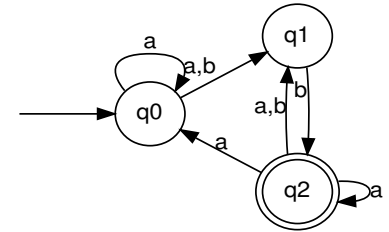
- Induction over the number of reduction steps n in $qu \Rightarrow^*_A pv$.
- For $n=0$: the claim holds trivially, $p'=q'$



Computing power of NFA

Proof (Lemma 1, cont.):

- For $n \rightarrow n+1$: Assume, the claim holds for all reductions of $\leq n$ steps.
- Let $qu \Rightarrow^*_A pv$ be a reduction of $n + 1$ steps. Then for some $q_1 \in Q$ and $u_1 \in T^*$ holds that $qu \Rightarrow_A q_1u_1 \Rightarrow^*_A pv$.
- Therefore, $\exists a \in T$, s.t. $u = au_1$ and $q_1 \in \delta(q, a)$.
- Since $\delta(q, a) \subseteq \delta'(q', a)$, for $q \in q'$, q'_1 can be chosen as $q'_1 = \delta'(q', a)$.
- Consequently, $q'u \Rightarrow_{A'} q'_1u_1$, where $q_1 \in q'_1$.
- By the induction assumption, $\exists p' \in Q'$, s.t. $q'_1u_1 \Rightarrow^*_{A'} p'v$ and $p \in p'$, which proves the claim. \square



Computing power of NFA

Proof (Theorem, cont.):

- Let $u \in L(A)$, i.e. $q_0u \Rightarrow^*_A p$, for some $q_0 \in Q_0$, $p \in F$.
- By Lemma 1, $\exists p'$, s.t. $q'_0u \Rightarrow^*_{A'} p'$, where $p \in p'$.
- By definition of F' , $p \in p'$ and $p \in F$ imply that $p' \in F'$, which proves $L(A) \subseteq L(A')$.
- For $L(A') \subseteq L(A)$, we prove Lemma 2.

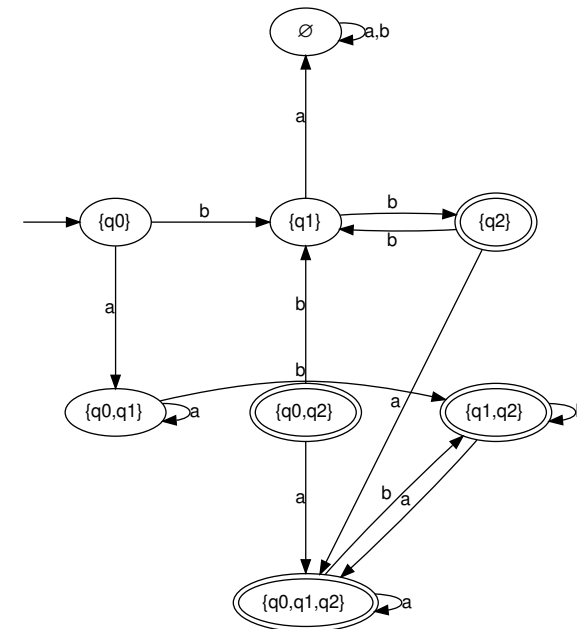
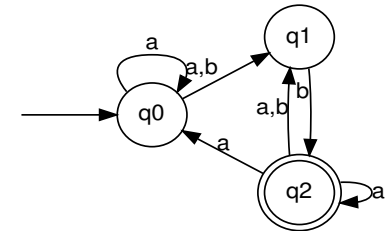
Computing power of NFA

Lemma 2:

- For all $p', q' \in Q', p \in Q$ and $u, v \in T^*$,
 - if $q'u \Rightarrow^*_{A'} p'v$ and $p \in p'$,
 - then $\exists q \in Q$, s.t.
 $qu \Rightarrow^*_{A} pv$ and $q \in q'$.

Proof:

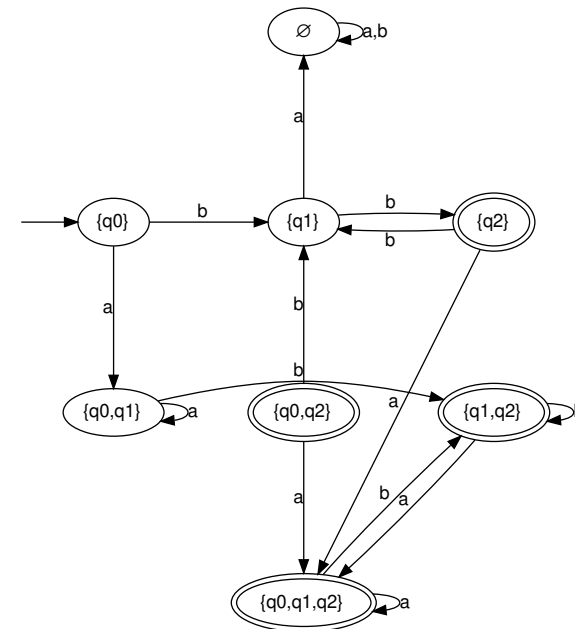
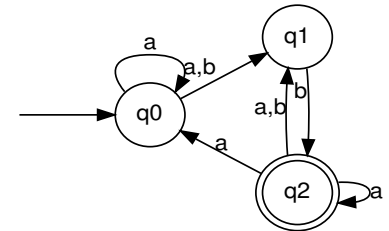
- Induction over the number of reduction steps n .
- For $n = 0$: The claim holds trivially.



Computing power of NFA

Proof (Lemma 2, cont.):

- For $n \rightarrow n + 1$: Assume, the claim holds for all reductions of $\leq n$ steps.
- Let $q'u \Rightarrow_{A'}^* p'v$ be a reduction of $n + 1$ steps. Then $q'u \Rightarrow_{A'}^* p'_1v_1 \Rightarrow_{A'} p'v$, where $v_1 = av$, for some $p'_1 \in Q'$ and $a \in T$.
- Then, $p \in p' = \delta'(p'_1, a) = \bigcup_{p_1 \in p'_1} \delta(p_1, a)$.
- Consequently, $\exists p_1 \in p'_1$, s.t. $p \in \delta(p_1, a)$.
- For this p_1 , it holds that $p_1v_1 \Rightarrow_A pv$.
- By the induction assumption, $qu \Rightarrow_{A'}^* p_1v_1$, for some $q \in q_0$, which implies the claim. \square



Computing power of NFA

Proof (Theorem, cont.):

- Let $q'_0 u \Rightarrow^*_{A'} p'$ and $p' \in F$.
- By the definition of F' , $\exists p \in p'$, s.t. $p \in F$.
- Then, by Lemma 2, for some $q_0 \in q'_0$, holds that $q_0 u \Rightarrow^*_A p$.
- This proves the claim of the theorem. \square

Corollaries

Corollary 1:

- The class of regular languages \mathcal{L}_3 is closed for the complement operation.

Proof:

- Let L be a language, recognized by a FA
 $A = (Q, T, \delta, q_0, F)$
- Then $\bar{L} = T^* - L$ can be recognized by a FA
 $A = (Q, T, \delta, q_0, Q - F)$

Corollaries

Corollary 2:

- The class of regular languages \mathcal{L}_3 is closed for the intersection operation.

Proof:

- We know, that \mathcal{L}_3 is closed for the union operation.
- $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.
- By Corollary 1, the claim follows.

FA – Myhill-Nerode Theorem

- Let L be a language over the alphabet T . The **relation E_L induced by language L** is a binary relation on T^* , for which it holds that
$$\forall u, v \in T^*, uE_Lv, \text{ if and only if } \nexists w \in T^*, \text{ s.t. exactly one of the words } uw \text{ and } vw \text{ is an element of } L$$
(i.e. $\forall w \in T^* : uw \in L$ if and only if $vw \in L$).
- E_L is an **equivalence relation** and it is **right-invariant**. (Right-invariant: if uE_Lv , then uwE_Lvw holds for every word $w \in T^*$.)
- The **index of the E_L** is the number of its equivalence classes.

Theorem (Myhill-Nerode): $L \subseteq T^*$ can be recognized by a deterministic FA if and only if E_L has a finite index.

FA – Myhill-Nerode Theorem

Theorem (Myhill-Nerode): $L \subseteq T^*$ can be recognized by a DFA if and only if E_L has a finite index.

- This index is equal to the number of states in the minimal DFA recognizing L .

DFA with minimum number of states

- The DFA A has a minimum number of states (**minimal DFA**), if there is no DFA A' , which recognizes the same language as A , but the number of states of A' is smaller than the number of state of A .

Theorem: The minimal DFA accepting the regular language L is unique, up to isomorphism.

DFA with minimum number of states

Theorem: The minimal DFA accepting the regular language L is unique, up to isomorphism.

- Let $A = (Q, T, \delta, q_0, F)$ be a DFA. Define a relation $R \subseteq Q \times Q$, s.t. pRq if \forall input word $x \in T^*$ it holds that $px \Rightarrow^*_A r$ if and only if $qx \Rightarrow^*_A r'$ for some $r, r' \in F$ states. ($r = r'$ is possible).
- States p and q are **distinguishable** if $\exists x \in T^*$, s.t. either $px \Rightarrow^*_A r, r \in F$, or $qx \Rightarrow^*_A r', r' \in F$, but both reductions are not possible. Otherwise, p and q are **indistinguishable**.
- If p and q are indistinguishable, then $\delta(p, a) = s$ and $\delta(q, a) = t$ are indistinguishable for any $a \in T$.
- If $\delta(p, a) = s$ and $\delta(q, a) = t$ are distinguishable for $x \in T^*$, then p and q are distinguishable also for ax .

DFA with minimum number of states

- Let $A = (Q, T, \delta, q_0, F)$ be a DFA. State q is **reachable** from the initial state if there is a reduction $q_0x \Rightarrow^* q$, where x is some word over T .
- The DFA $A = (Q, T, \delta, q_0, F)$ is **connected**, if all its states are reachable from the initial state.
- We define the **set H of reachabele states** as follows:
Let $H_0 = \{q_0\}$, $H_{i+1} = H_i \cup \{r \mid \delta(q, a) = r, q \in H_i, a \in T\}$, $i = 1, 2, \dots$
Then $\exists k \geq 0 : H_k = H_l$, for all $l \geq k$. Let $H = H_k$.
- We define the DFA $A' = (Q', T, \delta', q_0, F')$ with
 $Q' = H$, $F' = F \cap H$ and $\delta' : H \times T \rightarrow H$ s.t. $\delta'(q, a) = \delta(q, a)$,
if $q \in H$.
- It can be shown that A' is connected and accepts the same language as A . A' is the **largest connected subautomaton** of A .

DFA with minimum number of states

Computing_Reachable_States

(from: https://en.wikipedia.org/wiki/DFA_minimization)

- let reachable_states := {q0}
- let new_states := {q0}
- do {
- temp := the empty set
- for each q in new_states do
- for each c in T do
- temp := temp \cup {p such that p = $\delta(q,c)$ }
- new_states := temp \setminus reachable_states
- reachable_states := reachable_states \cup new_states
- } while (new_states \neq the empty set)
- unreachable_states := Q \setminus reachable_states

DFA with minimum number of states

- Computing the minimal DFA (Hopcroft's partition refinement):
 - Determine whether the automaton is connected or not.
 - If it is not connected, then consider the largest connected subautomaton.
In the following, we assume, that the DFA is connected.
 - Partition the set of states according to distinguishability (states are divided into equivalence classes) (Steps 1-3)

DFA with minimum number of states

- **Step 1:**

- Divide the set of states into two partitions: F and $Q - F$. (The states in F can be distinguished from the states in $Q - F$ by the empty word).
- Repeat splitting of the partitions (Step 2) into additional partitions as long as the number of partitions remains the same.

- **Step 2:**

- Consider an arbitrary partition P of states. Take an input symbol a and consider $\delta(p, a)$ for each state $p \in P$. If the obtained states belong to different partitions, then split P into as many new partitions as arising in this way.
- Perform this procedure for each input symbol and each partition, until no new partition is created.

DFA with minimum number of states

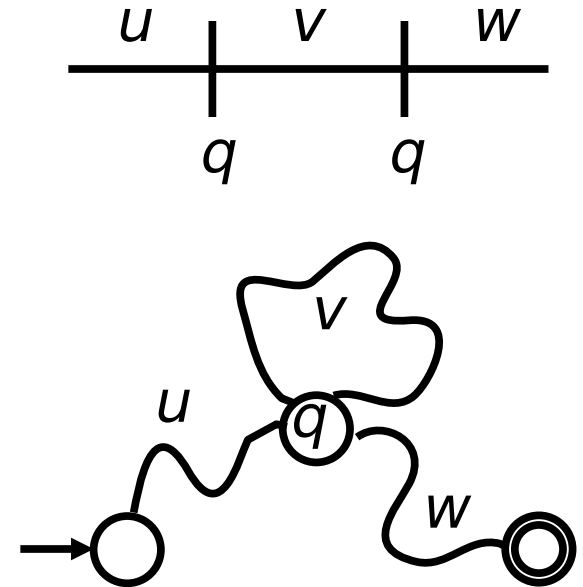
- **Step 3:**
 - Determine the DFA with the minimum number of states:
 - For each partition B_i , consider a representative state b_i .
 - Construct a DFA $A' = (Q', T, \delta', q_0, F')$, where
 - Q' is set of representatives of the partitions,
 - q'_0 is the representative of the partition containing q_0 ,
 - $\delta'(b_i, a) = b_j$, if $\exists q_i \in B_i$ and $q_j \in B_j$, s.t. $\delta(q_i, a) = q_j$.
 - $F' = \{b_f\}$ is the representative of the partition that contains the elements of F .

Pumping lemma for regular languages

- A necessary condition for regular languages (i.e. recognizable by a FA).
- **Theorem** (pumping lemma for regular languages):
For every regular language L there exists a natural number n , s.t. for all words $z \in L$ with $|z| > n$, holds that z can be written as $z = uvw$, satisfying the following conditions:
 1. $|uv| \leq n$,
 2. $|v| > 0$,
 3. $uv^i w \in L$, for all $i \geq 0$.

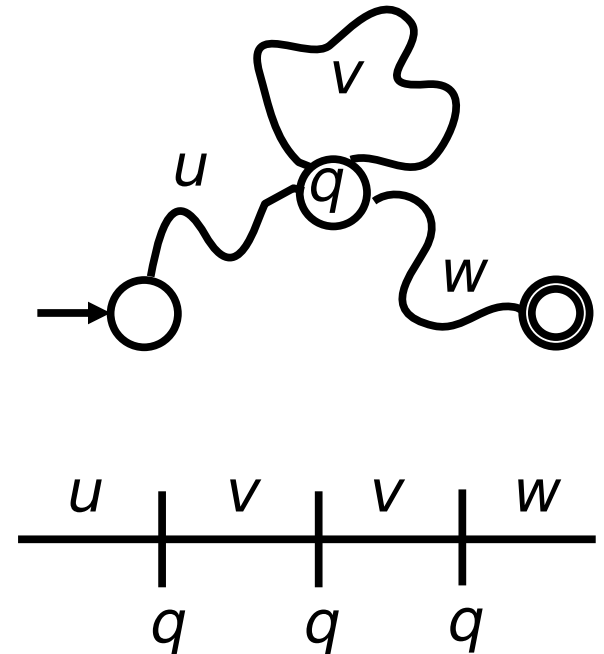
Pumping lemma for regular languages

- **Proof.:**
 - Let L be a regular language and $A=(Q, T, \delta, q_0, F)$ be a minimal DFA, s.t. $L(A)=L$.
 - Let $n=|Q|+1$. Let $z \in L$ be an arbitrary word with $|z|>n$.
 - Consider A with input z . There must be a state q that A visits at least twice during the processing of z . Such a state q must already exist during the first n state transitions.
 - Let u be the prefix of z processed by A up to the first occurrence of q , and let v be the subword of z processed between the first and second occurrences of q . Then $|uv| \leq n$.



Pumping lemma for regular languages

- **Proof** (cont.):
 - Since at least one state transition has occurred in A between two occurrences of q , i.e. at least one symbol has been read, therefore $|v| > 0$.
 - If A starts from the state q and reads the word w , it reaches the accepting end state. Accordingly, A accepts uw .
 - Similarly, A accepts all words of the form $uv^i w$, $i > 0$, since after reading u , A goes to state q , starting from q after reading v^i , A returns to state q , finally after reading w , A reaches an accepting end state. This completes the proof. \square



Application of the pumping lemma

Claim: The language $L = \{a^j b^j \mid j \geq 1\}$ is not regular.

Proof: Assume that G is a regular grammar generating L .

Then, by the regular pumping lemma,

$\exists n \geq 0$, s.t. $\forall z \in L$ words with $|z| > n$,

z can be written as $z = uvw$, satisfying

$|uv| \leq n$, $|v| > 0$, and $uv^i w \in L$, for all $i \geq 0$.

Consider a word $a^m b^m$, where $m > n$.

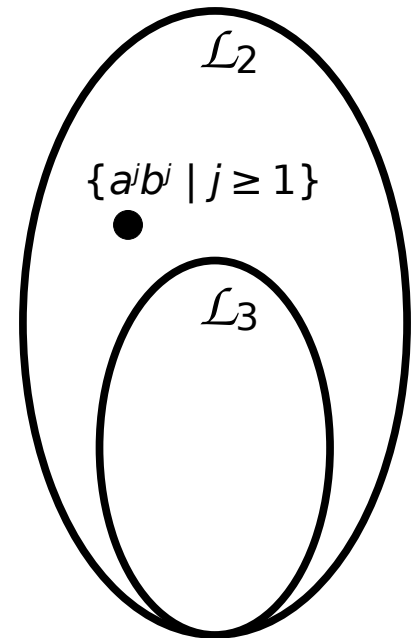
Since $|uv| \leq n$, uv contains a symbols.

Since $|v| > 0$, for $i \geq 2$, $uv^i w$ contains more

a symbols than b symbols.

Consequently, $uv^i w \notin L$.

□



A context-free grammar generating L :

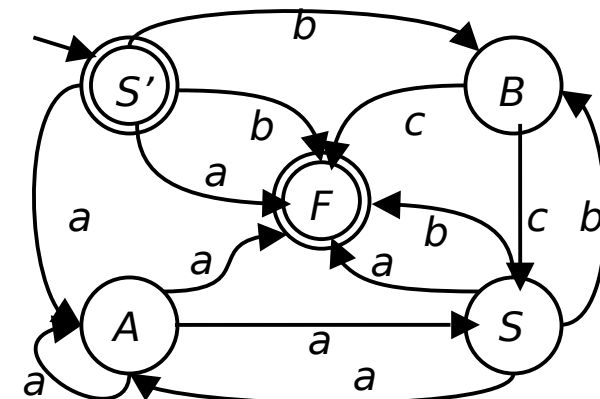
$S \rightarrow ab, S \rightarrow aSb$.

Transforming Regular Grammars to Equivalent FA

- 1) Construct an ε -free regular grammar G' from G (see next slide);
- 2) Create a FA M , with a state for every non-terminal in G' . Set the state representing the start symbol S' in G' to be the start state;
- 3) Add a new state F , which is final state;
- 4) If the production $S' \rightarrow \varepsilon$ is in G' ,
 - set the state representing S' to be final state;
- 5) For every production $A \rightarrow aB$ in G' ,
 - add a transition from state A to state B labelled with a ;
- 6) For every production $A \rightarrow a$ in G' ,
 - add a transition from A to the final state F .

Example:

- G :
 - $S \rightarrow a|aA|bB|\varepsilon$
 - $A \rightarrow aA|aS$
 - $B \rightarrow cS|\varepsilon$
- G' :
 - $S' \rightarrow a|b|aA|bB|\varepsilon$
 - $S \rightarrow a|b|aA|bB$
 - $A \rightarrow a|aA|aS$
 - $B \rightarrow c|cS$



Making a Regular Grammar ε -Free

A regular grammar G is ε -free if it has no ε -productions except for $S \rightarrow \varepsilon$, where S is the start symbol, and S does not appear on the right hand side of the production rules.

Making a regular grammar G ε -free:

- 1) Copy all non ε -productions from G to G' .
Let S be the start symbol in G' ;
- 2) For any non-terminal N which can become ε (While $\exists N : N \rightarrow \varepsilon$ is a production do),
 - copy every rule in which N appears on the right hand side both with and without N ;
- 3) If $S \rightarrow \varepsilon$ was in the original set of rules,
 - add a new start symbol S' in G' ,
 - add the rule $S' \rightarrow \varepsilon$ and
 - copy all the production rules with S on the left hand side to ones with S' on the left hand side.

Example:

- G :
 - $S \rightarrow aA \mid bB \mid \varepsilon$
 - $A \rightarrow aA \mid a \mid \varepsilon$
 - $B \rightarrow bB \mid b \mid \varepsilon$
- 1)
 - $S \rightarrow aA \mid bB$
 - $A \rightarrow aA \mid a$
 - $B \rightarrow bB \mid b$
- 2)
 - $S \rightarrow aA \mid bB \mid a \mid b$
 - $A \rightarrow aA \mid a$
 - $B \rightarrow bB \mid b$
- 3)
 - $S' \rightarrow aA \mid bB \mid a \mid b \mid \varepsilon$
 - $S \rightarrow aA \mid bB \mid a \mid b$
 - $A \rightarrow aA \mid a$
 - $B \rightarrow bB \mid b$

Transforming FA to Regular Grammar

Transforming FA A to a regular grammar G :

- 1) Let T be the terminal alphabet of the grammar G – the same as that of A .
- 2) The set of non-terminals in G is set to be Q – the set of states of A .
- 3) The start state S of A will be the start symbol of G be.
- 4) Initially, let the set of rules in G be \emptyset
For every transition $(q,a) \rightarrow q'$ of A ,
 - a) add the production $q \rightarrow aq'$;
 - b) if q' is a final state also add the production $q \rightarrow a$.
- 5) If S is a final state of A add the production rule $S \rightarrow \varepsilon$.
- 6) If grammar is not ε -free, make it ε -free (see previous slide).

Example:

- $A = (Q, T, \delta, S, \{S, C\})$ with δ :
 - $(S, a) \rightarrow A$,
 - $(S, b) \rightarrow B$,
 - $(A, a) \rightarrow B$,
 - $(A, a) \rightarrow C$,
 - $(B, b) \rightarrow A$,
 - $(B, b) \rightarrow C$,
 - $(C, c) \rightarrow C$.
- 4)
 - $S \rightarrow aA \mid bB$,
 - $A \rightarrow aB \mid aC \mid a$,
 - $B \rightarrow bA \mid bC \mid b$,
 - $C \rightarrow cC \mid c$.
- 5)
 - $S \rightarrow aA \mid bB \mid \varepsilon$
 - $A \rightarrow aB \mid aC \mid a$
 - $B \rightarrow bA \mid bC \mid b$
 - $C \rightarrow cC \mid c$.

Transforming FA to Regular Expression

Idea: Assume, states of FA A are enumerated: $1, \dots, n$, start state: 1.

We compute regular expressions $T(i, j, k)$ that describe all strings that take us from state i to j through states $\{1, 2, \dots, k\}$. The language $L(A)$ is the union of all strings that take us from state 1 to a final state $f \in F$ through any state:

$$L(A) = \bigcup_{f \in F} T(1, f, n) .$$

Calculating $T(i, j, k)$

1) Base case, $k=0$:

a) If $i=j$: $T(i, i, 0) = \varepsilon + a + \dots + z$, where a to z are the labels on transition arcs going from state i to itself.

If no such arcs exist, $T(i, i, 0) = \varepsilon$.

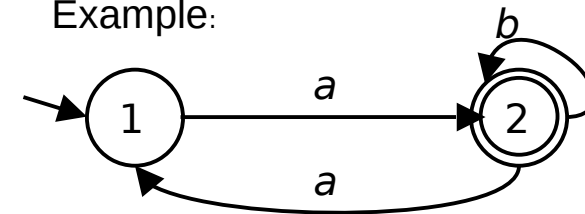
b) If $i \neq j$: $T(i, j, 0) = a + \dots + z$, where a to z are the labels on transition arcs going from state i to state j

If no such arcs exist, $T(i, j, 0) = \emptyset$.

2) Inductive case, $k > 0$:

$$T(i, j, k) = T(i, j, k-1) + T(i, k, k-1)(T(k, k, k-1))^*T(k, j, k-1)$$

Example:



- $T(1, 1, 0) = \varepsilon$
 $T(2, 2, 0) = \varepsilon + b$
 $T(1, 2, 0) = a$
 $T(2, 1, 0) = a$
- $T(1, 1, 1) = \varepsilon + \varepsilon(\varepsilon)^* \varepsilon = \varepsilon$
 $T(2, 2, 1) = \varepsilon + b + a(\varepsilon)^* a = \varepsilon + b + aa$
 $T(1, 2, 1) = a + \varepsilon(\varepsilon)^* a = a$
 $T(2, 1, 1) = a + a(\varepsilon)^* \varepsilon = a$
- $T(1, 1, 2) = \dots$
 $T(2, 2, 2) = \dots$
 $T(1, 2, 2) =$
 $a + a(\varepsilon + b + aa)^*(\varepsilon + b + aa) =$
 $a + a(\varepsilon + b + aa)^+ =$
 $a + a(b + aa)^* =$
 $\underline{a(b + aa)^*}$
 $T(2, 1, 2) = \dots$

Transforming Regular Expression into an NFA

Transforming Regular Expression R
into a NFA N :

1. If $R = a$, for $a \in T$, then $L(R) = \{a\}$

2. If $R = \varepsilon$, then $L(R) = \{\varepsilon\}$

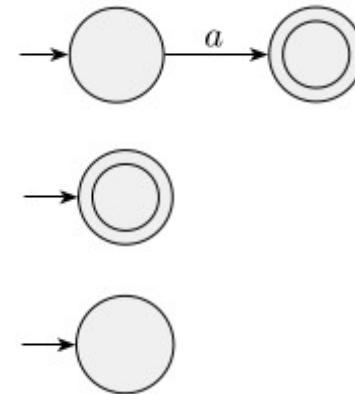
3. If $R = \emptyset$. Then $L(R) = \emptyset$

4. $R = R_1 \cup R_2$

5. $R = R_1 \cdot R_2$

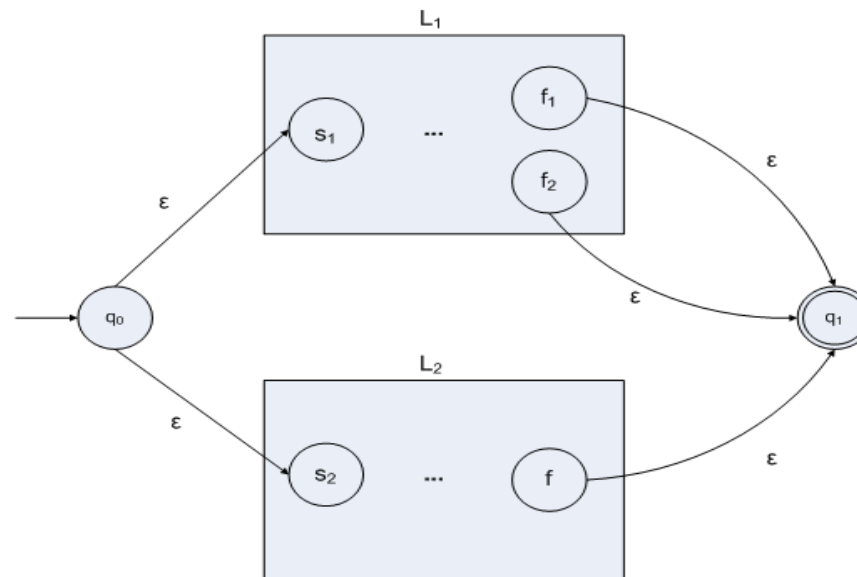
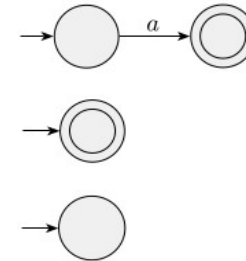
6. $R = R_1^*$

NFA recognizes $L(R)$



Transforming Regular Expression into an NFA

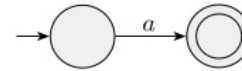
1. If $R = a$, for $a \in T$, then $L(R) = \{a\}$
2. If $R = \varepsilon$, then $L(R) = \{\varepsilon\}$
3. If $R = \emptyset$. Then $L(R) = \emptyset$
4. $R = R_1 \cup R_2$



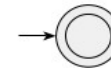
5. $R = R_1 \cdot R_2$
6. $R = R_1^*$

Transforming Regular Expression into an NFA

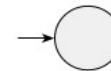
1. If $R = a$, for $a \in T$, then $L(R) = \{a\}$



2. If $R = \varepsilon$, then $L(R) = \{\varepsilon\}$

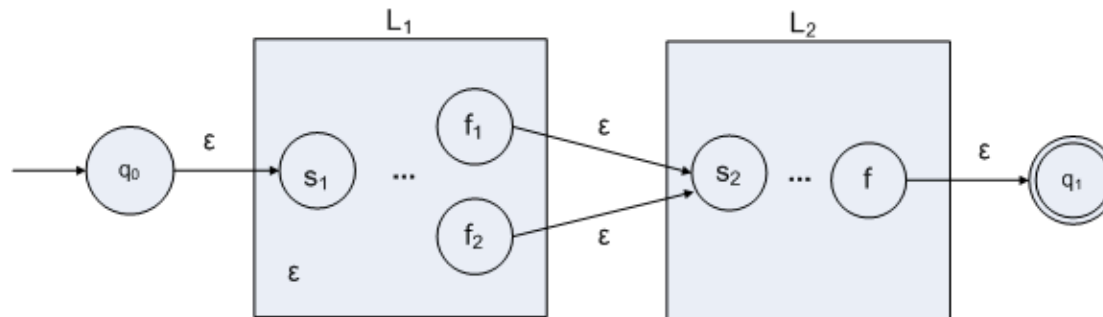


3. If $R = \emptyset$. Then $L(R) = \emptyset$



4. $R = R_1 \cup R_2$

5. $R = R_1 \cdot R_2$



6. $R = R_1^*$

Transforming Regular Expression into an NFA

1. If $R = a$, for $a \in T$, then $L(R) = \{a\}$
2. If $R = \varepsilon$, then $L(R) = \{\varepsilon\}$
3. If $R = \emptyset$. Then $L(R) = \emptyset$
4. $R = R_1 \cup R_2$
5. $R = R_1 \cdot R_2$
6. $R = R_1^*$

