1.2 Regular languages (type 3)

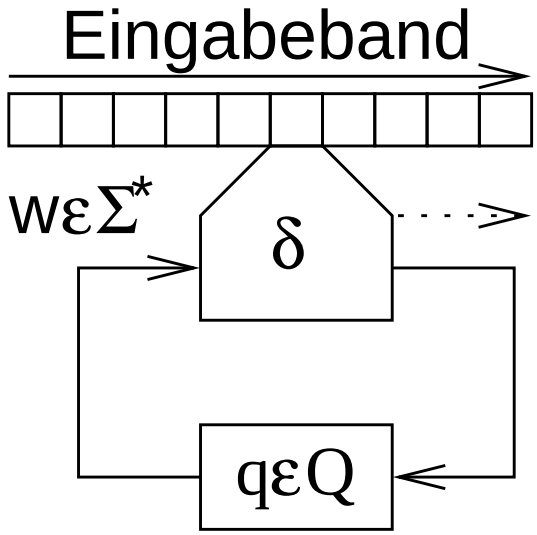
- ⇝ (Non)deterministic finite automata
- ⇝ Regular expressions
- Non regular languages
- Minimal automata
- Decidable properties

1.2.1 (Deterministic) finite automata

A deterministic finite automaton consists from:

(a finite acceptor or deterministic finite automaton=DFA)

- Q, a finite set on states;
- Σ, a finite set of (input) symbols, (alphabet);
- $\delta : Q \times \Sigma \rightarrow Q$, the transition function;
- $s \in Q$, the initial state;
- $F \subseteq Q$, the set of final states.



How does a finite automaton work?

We extend the definition of δ for words:

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

$\hat{\delta}(q,aw) = \hat{\delta}(\delta(q,a),w)$ — a transition for the input symbols

$A = (Q,\Sigma,\delta,s,F)$ accepts the language

$L(A) := \left\{ w \in \Sigma^* : \hat{\delta}(s,w) \in F \right\}$ (Schöning: $T(A)$)

Equivalent definition:

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

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

Proof: Exercise. Show that $\forall q,u,v : \hat{\delta}(q,uv) = \hat{\delta}(\hat{\delta}(q,u),v)$.

Graph interpretation

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

$G_A = (Q, E)$,

every $e = (q, q') \in E$ is labelled $\ell(e) = a$ if $q' = \delta(q, a)$

Multi-graph!

Lemma:

$\forall w \in \Sigma^* : w \in L(A) \Leftrightarrow$

$\exists$path $P = sq_1q_2 \cdots f = s \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} f$

where $f \in F, w = a_1a_2 \cdots a_k$.

Proof: Exercise

Terminology:

When we mention a path in A, we mean a path in G_A.

Notation for path

$P = sq_1q_2 \cdots f$ a sequence of nodes

$P = s \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} f$ a sequence of (direct) transition

$P = s \overset{w}{\Rightarrow} f$, where $w = a_1 \cdots a_k$. P is by w labelled

$q \overset{*}{\Rightarrow} r$ there is a path from q to r, i.e. r is reachable from q

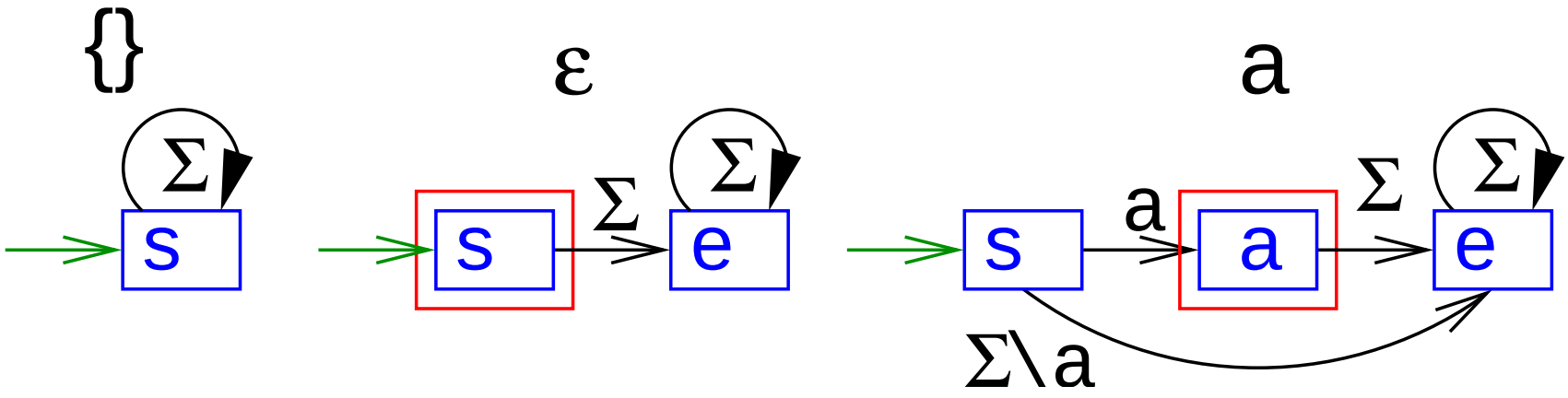
By definition $s \overset{*}{\Rightarrow} s$ (reflexivity)

Duality: Grammars $\leftrightarrow$ Machines

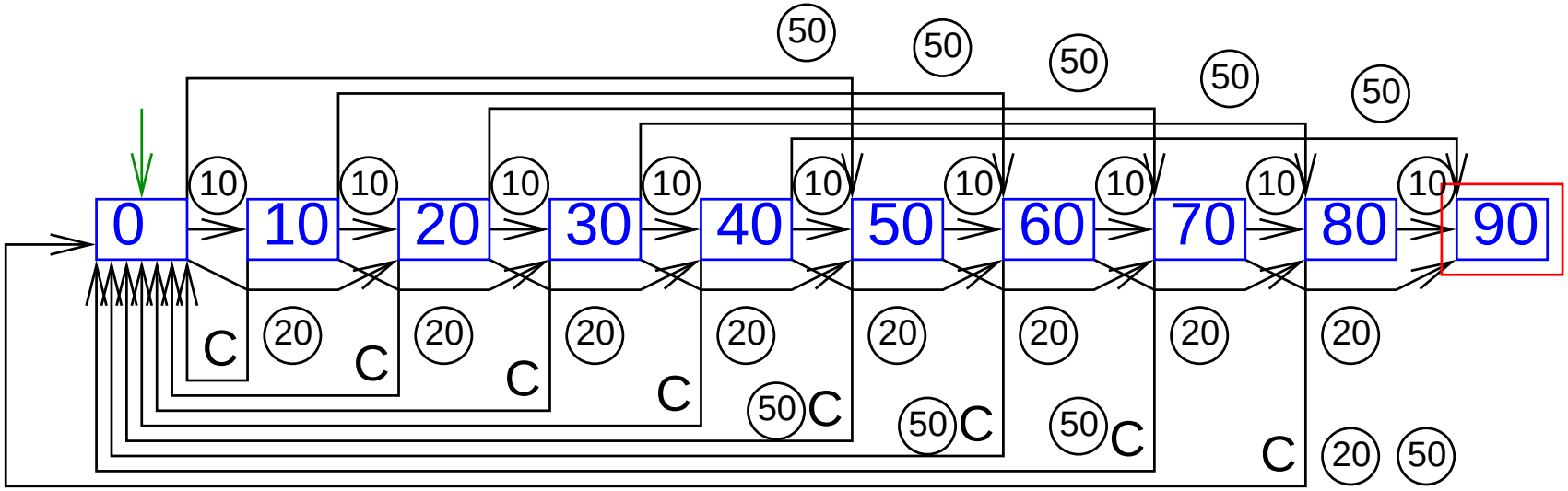
Grammars generate words.

Machines accept/recognize/parse words.

Finite automata: A simple example



Example: Ticket automaton



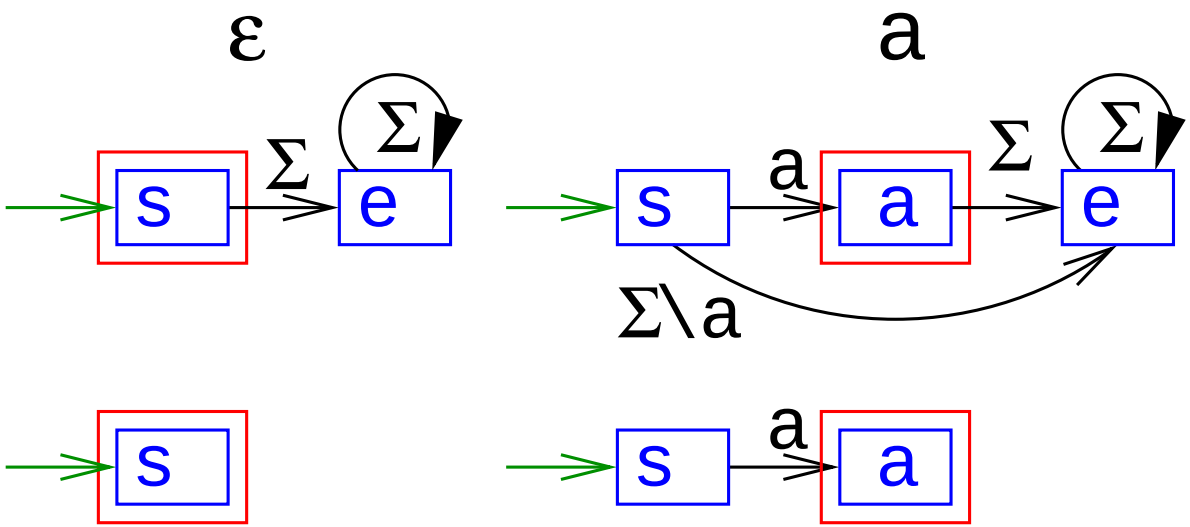
Completion

Often we do not give all values of δ.

Convention: There is always an error state e so that

$\delta(q,c) = e$ when we could not accept more symbols.

$\delta(e,c) = e \forall c \in \Sigma$



Proposition:

The languages accepted by a DFA are of Chomsky type 3

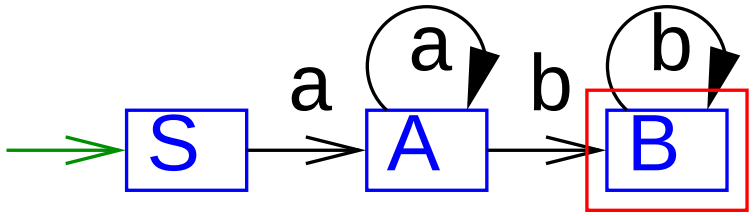
Let $A = (Z, \Sigma, \delta, S, F)$ be a DFA.

Consider the grammar $G = (Z, \Sigma, P, S)$, where

$$P = \{Q \to aQ' : \delta(Q,a) = Q'\} \cup$$
$$\{Q \to a : \delta(Q,a) = Q' \in F\} \cup$$
$$\{S \to \varepsilon : S \in F\} \quad .$$

then $L(G) = L(A)$ (ε is eleminated. . .)

Example: $\{a^n b^m : n \geq 1, m \geq 1\}$



$G = (\{S, A, B\}, \{a, b\}, P, S)$

q	c	$\delta(q,c)$	$\in P$
S	a	A	$S \to aA$
A	a	A	$A \to aA$
A	b	B	$A \to bB, A \to b$
B	b	B	$B \to bB, B \to b$

What about $\delta(S, b)$?

All recognizable languages by a DFA are of Chomsky type 3

Let $A = (Z, \Sigma, \delta, S, F)$ be a DFA.

Consider the grammar $G = (Z, \Sigma, P, S)$, where

$$P = \{Q \to aQ' : \delta(Q,a) = Q'\} \cup$$
$$\{Q \to a : \delta(Q,a) = Q' \in F\} \cup$$
$$\{S \to \varepsilon : S \in F\} \quad .$$

Then $L(G) = L(A)$.

Idea: $\exists$ an 1-1 relation between

the derivations $S \Rightarrow w_1 A_1 \Rightarrow w_1 w_2 A_2 \Rightarrow \cdots \Rightarrow w_1 w_2 \cdots w_{n-1} A_{n-1} \Rightarrow w$ an

the DFA computation paths $S \overset{w_1}{\Rightarrow} A_1 \overset{w_2}{\Rightarrow} A_2 \overset{w_3}{\Rightarrow} \cdots \overset{w_n}{\Rightarrow} f \in F$.

Proof: $L(G) = L(A)$:

If $w = \varepsilon$**:** $\varepsilon \in L(G) \Leftrightarrow S \to \varepsilon \in P \Leftrightarrow S \in F \Leftrightarrow \varepsilon \in L(A)$

$A = (Z, \Sigma, \delta, S, F)$, $G = (Z, \Sigma, P, S)$, where $P = \{Q \to aQ' : \delta(Q,a) = Q'\}$
$\{Q \to a : \delta(Q,a) = Q' \in F\} \cup \{S \to \varepsilon : S \in F\}$

Proof (sketch) $L(G) = L(A)$ **If** $|w| = n$**,** $n > 0$:

$w_1 \cdots w_n \in L(G)$

$\Leftrightarrow S \Rightarrow w_1 A_1 \Rightarrow w_1 w_2 A_2 \stackrel{*}{\Rightarrow} w_1 \cdots w_{n-1} A_{n-1} \Rightarrow w_1 \cdots w_n$

$\Leftrightarrow \{S \to w_1 A_1, A_1 \to w_2 A_2, \ldots, A_{n-2} \to w_{n-1} A_{n-1}, A_{n-1} \to w_n\} \subseteq P$

$\Leftrightarrow \delta(S, w_1) = A_1, \delta(A_1, w_2) = A_2, \ldots, \delta(A_{n-1}, w_n) = A_n \in F$

$\Leftrightarrow \exists$computation path $S \stackrel{w_1}{\Rightarrow} A_1 \stackrel{w_2}{\Rightarrow} A_2 \stackrel{w_3}{\Rightarrow} \cdots A_{n-1} \stackrel{w_n}{\Rightarrow} A_n \in F$

$\Leftrightarrow w_1 \cdots w_n \in L(A)$

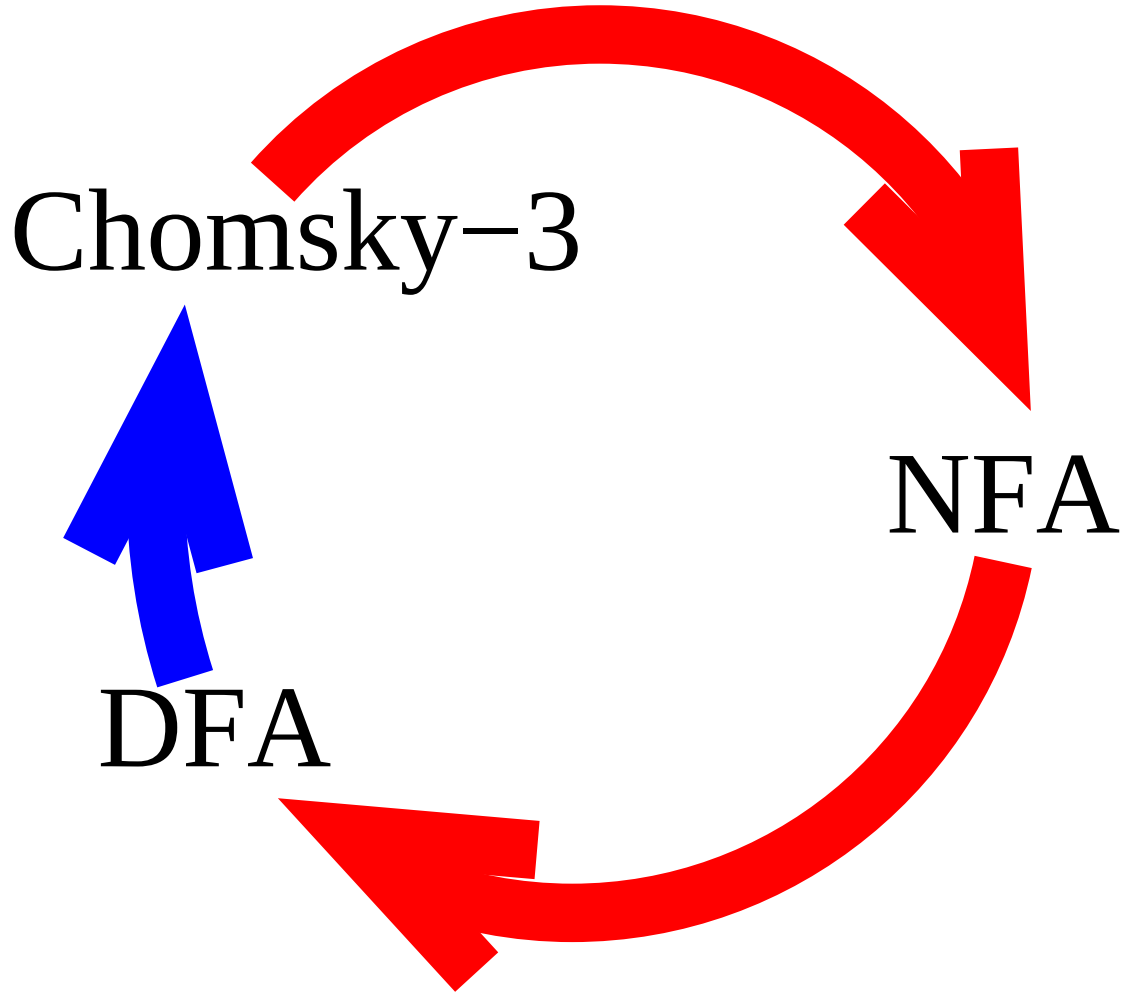
(In 2 directions '$\Leftrightarrow$' always establish) ■

$A = (Z, \Sigma, \delta, S, F)$, $G = (Z, \Sigma, P, S)$, where $P = \{Q \to aQ' : \delta(Q, a) = Q'\}$
$\{Q \to a : \delta(Q, a) = Q' \in F\} \cup \{S \to \varepsilon : S \in F\}$

Proposition: Chomsky type 3 languages are by finite automata recognizable.

(The reverse proposition will be in section 1.2.2)

The Scheme



1.2.2 Nondeterministic finite automata NFA

- more than 1 transitions are allowed from a given state

Nondeterministic finite automaton A

- Q, a set of states
- Σ, an alphabet
- $\delta: Q \times \Sigma \rightarrow 2^Q$, transition function
- $s \in Q$, input state
- $F \subseteq Q$, final states

The transition from q to q' by an input a: $q' \in \delta(q,a)$

more possibilities!

Nondeterministic finite automaton A

Variant (equivalent) — δ as a relation.

- Q, a set of states
- Σ, an alphabet
- $\delta \subseteq Q \times \Sigma \times Q$ transition relation
- $s \in Q$, input state
- $F \subseteq Q$, final states

A could make a transition from q to q' when $a \in \Sigma$ and $(q, a, q') \in \delta$.

Extension of δ

Subset of states: $\overline{\delta} : 2^Q \times \Sigma \to 2^Q$

$$\overline{\delta}(M,a) := \bigcup_{p \in M} \delta(p,a)$$

Subset of states and an input word : $\hat{\delta} : 2^Q \times \Sigma^* \to 2^Q$

$$\hat{\delta}(M,\varepsilon) := M$$
$$\hat{\delta}(M,aw) := \hat{\delta}(\overline{\delta}(M,a),w)$$

$$L(A) := \left\{ w \in \Sigma^* : \hat{\delta}(\{s\},w) \cap F \neq \emptyset \right\}$$

(Multi)Graph interpretation for $L(A)$

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

$G_A = (Q, E)$ Function interpretation for δ

every $e = (q, q') \in E$ is labelled $\ell(e) = a$ if $q' \in \delta(q, a)$

$G_A = (Q, \delta)$ Relation interpretation for δ

label the arc e by $\ell(e) = a$ as (q, a, q').

„Multi“=parallel arcs are allowed (different labels)

$w \in L(A) \Leftrightarrow \exists$ path $P = s \overset{a_1}{\Rightarrow} q_1 \overset{a_2}{\Rightarrow} \cdots \overset{a_k}{\Rightarrow} f$ in A (in G(A)) :

$$f \in F \wedge w = a_1 a_2 \cdots a_k$$

In words: A path from s to a final state is labelled by w.

Lemma: $\hat{\delta}(M,w) = \left\{ q \in Q : \exists p \in M : p \stackrel{w}{\Rightarrow} q \right\}$

Proof by induction on $|w|$:

$\hat{\delta}(M,\varepsilon) = M$

$n \rightsquigarrow n+1$:

$\hat{\delta}(M,aw) = \hat{\delta}(\overline{\delta}(M,a),w)$

$= \left\{ r \in Q : \exists q \in \overline{\delta}(M,a) : q \stackrel{w}{\Rightarrow} r \right\}$ (IH)

$= \left\{ r \in Q : \exists p \in M, q \in \delta(p,a) : q \stackrel{w}{\Rightarrow} r \right\}$ (Def. $\overline{\delta}$)

$= \left\{ r \in Q : \exists p \in M : p \stackrel{aw}{\Rightarrow} r \right\}$ (Graph interpretation.)

□

Corollary: $L(A) = \left\{ w \in \Sigma^* : \exists z \in F : s \stackrel{w}{\Rightarrow} z \right\}$

NFA→DFA

Given: NFA $A = (Q, \Sigma, \delta, s, F)$

Proposition: (Subset-construction)

[Rabin, Scott 1959, IBM J. of R&D]

DFA $A' := (2^Q, \Sigma, \overline{\delta}, \{s\}, \{M \subseteq Q : M \cap F \neq \emptyset\})$ accepts $L(A)$.

Exercise: Find an algorithm for a given NFA A and a word w that computes $\hat{\delta}(\{s\}, w)$ in time $\mathcal{O}(|w| \cdot |\delta|)$. Here $|\delta|$ is the numbers of the transitions of the form $p \in \delta(q, a)$, enough to define δ.

Subset-construction

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

$A' := (2^Q, \Sigma, \overline{\delta}, \{s\}, F')$, $F' := \{M \subseteq Q : M \cap F \neq \emptyset\}$, where

$\overline{\delta}(M,a) := \bigcup_{p \in M} \delta(p,a)$

Statement: $L(A') = L(A)$

Proof: First prove by induction on w that $\hat{\overline{\delta}}(\{s\}, w) = \hat{\delta}(\{s\}, w)$.

Then

$$L(A) = \left\{ w \in \Sigma^* : \hat{\delta}(\{s\}, w) \cap F \neq \emptyset \right\} \qquad \text{Def. L(A)}$$

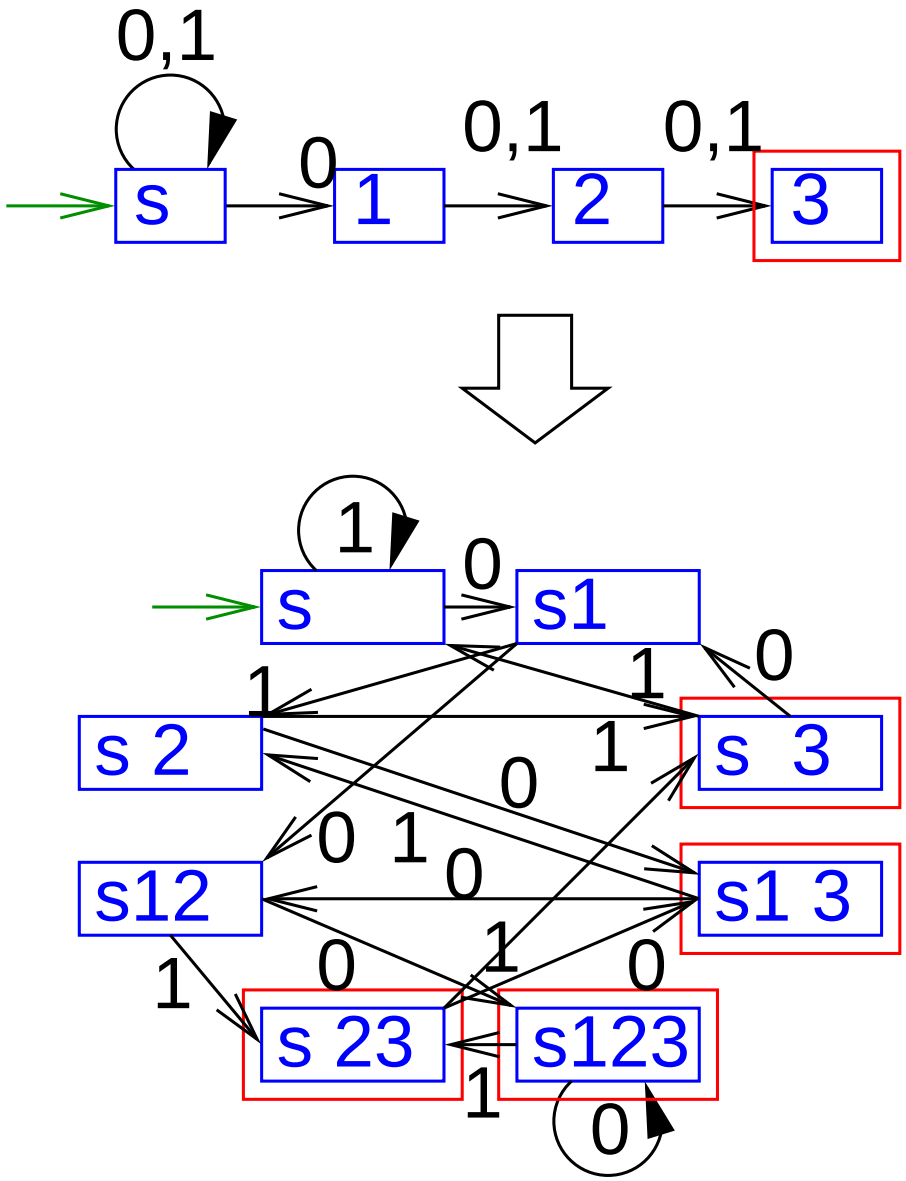
$$= \left\{ w \in \Sigma^* : \hat{\delta}(\{s\}, w) \in F' \right\} \qquad \text{Def. F'}$$

$$= L(A') \qquad \text{Def. L(A')}$$

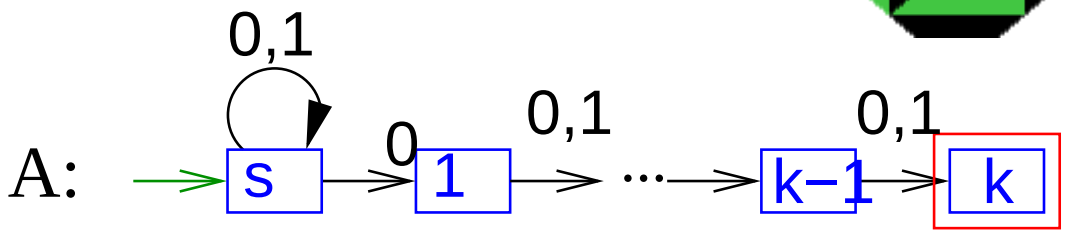
($\hat{\delta}$ dances here in two weddings!) $\square$

Example

q	$\overline{\delta}(q,0)$	$\overline{\delta}(q,1)$
s	$s,1$	s
$s,1$	$s,1,2$	$s,2$
$s,2$	$s,1,3$	$s,3$
$s,3$	$s,1$	s
$s,1,2$	$s,1,2,3$	$s,2,3$
$s,1,3$	$s,1,2$	$s,2$
$s,2,3$	$s,1,3$	$s,3$
$s,1,2,3$	$s,1,2,3$	$s,2,3$



More general example



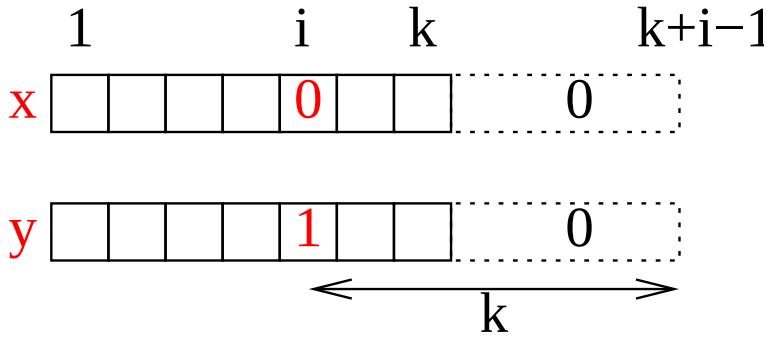
Proposition: $\nexists \text{DFA } A' = (Q, \Sigma, \delta, s, F) : L(A') = L(A) \wedge |Q| < 2^k$

Proof: Assume: $\exists A'$

$\longrightarrow \exists x \neq y \in \{0,1\}^k : \hat{\delta}(s,x) = \hat{\delta}(s,y)$ (Pigeon principle)

where i: $x[i] \neq y[i]$,

Let $x[i] = 0, y[i] = 1$



then $x0^{i-1} \in L(A)$

and $y0^{i-1} \notin L(A)$.

But, $\hat{\delta}(s, x0^{i-1}) = \hat{\delta}(\hat{\delta}(s,x), 0^{i-1})$

$= \hat{\delta}(\hat{\delta}(s,y), 0^{i-1}) = \hat{\delta}(s, y0^{i-1})$.

So either both $x0^{i-1}$ and $y0^{i-1}$ are accepted or both not accepted.

A contradiction. □

Implementing hints

Consider only the subsets accessible from $\{s\}$:

$Q' := \{\{s\}\}$ // states of A'

Queue todo:= Q'

while $\exists M \in$todo **do**

todo:= todo$\setminus M$

foreach $a \in \Sigma$ **do**

if $M' = \overline{\delta}(M,a) \notin Q'$ **then**

insert M' into Q'

insert M' into todo

Often $|Q'| \ll 2^{|Q|}$!

Type-3→NFA

Let $G = (V, \Sigma, P, S)$ be a type 3 grammar.

Consider the NFA $A = (V \cup \{f\}, \Sigma, \delta, S, \{f\} \cup \{S : S \to \varepsilon \in P\})$,

where

$$\delta = \{(q, a, q') : q \to aq' \in P\} \cup$$
$$\{(q, a, f) : q \to a \in P\}$$

(Relation notation for δ).

There is a 1-1 relation between the derivations of the form

$S \Rightarrow w_1 A_1 \Rightarrow w_1 w_2 A_2 \Rightarrow \cdots \Rightarrow w_1 w_2 \cdots w_{n-1} A_{n-1} \Rightarrow w$ in G and

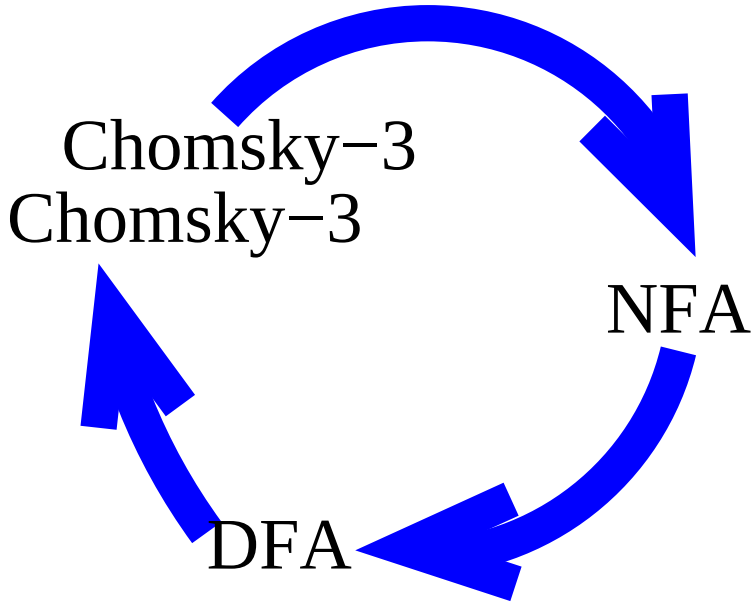
accepted paths of the form

$S \xrightarrow{w_1} A_1 \xrightarrow{w_2} A_2 \xrightarrow{w_3} \cdots \xrightarrow{w_n} f$ by A.

Hence $L(A) = L(G)$. ■

Unambiguous grammars of type-3

Proposition: $\forall L \in$ type-3 $: \exists$ type-3 grammar with unambiguous derivations.



Proof (sketch): Let A DFA and $L(A) = L$.

The corresponding type-3 grammar for A has unambiguous derivations.

1.2.3 Regular expressions

A regular (?) expression describes a regular (?) language .

expression	describes	remark
$\emptyset$	$\emptyset$	
ε	$\{\varepsilon\}$	
a	$\{a\}$	$a \in \Sigma$
$\alpha \cup \beta$	$L(\alpha) \cup L(\beta)$	α describes $L(\alpha)$ (Synonym: $\alpha \mid \beta$)
$\alpha \cdot \beta$	$L(\alpha) \cdot L(\beta)$	β describes $L(\beta)$
(α)	$L(\alpha)$	
α^*	$L(\alpha)^*$	
α^+	$L(\alpha)^+$	

Conventions: '$\cdot$' dropped, $L(\cdot)$ dropped

Syntax (reg. expression) versus semantic (reg. languages)

A computer program processes by syntactic objects.

Programm verification

We prove from the semantic point, that the object will be correctly processed.

Example

- the digit before the last 0: $(0 \cup 1)^* 0 (0 \cup 1)$
- contains 10: $(0 \cup 1)^* 10 (0 \cup 1)^*$
- does not contain 10: $0^* 1^*$
- contains 101: $(0 \cup 1)^* 101 (0 \cup 1)^*$
- does not contain 101: $0^* 1^* \cup (0^* 1^* 100)^* 0^* 1^* 10 (\varepsilon \cup 00^* 1^*)$
- all integers: $(\varepsilon \cup + \cup -)(1 \cup \cdots \cup 9)(0 \cup \cdots \cup 9)^* \cup 0$

Theorem: Regular expression $\Leftrightarrow$ type-3 languages

[Kleene]

The Scheme

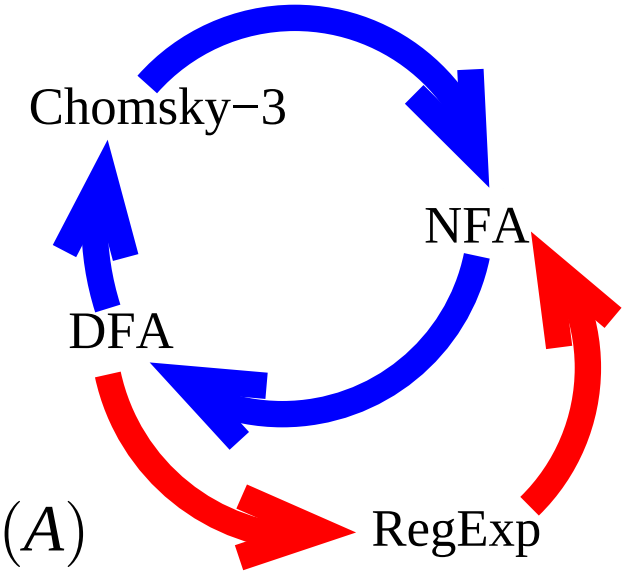
to prove:

$\rightarrow$ regular expression α

$\rightsquigarrow$ NFA A with $L(\alpha) = L(A)$

$\leftarrow$ DFA A

$\rightsquigarrow$ regular expression α with $L(\alpha) = L(A)$



Regular expression $\longrightarrow$ NFA

Proof idea:

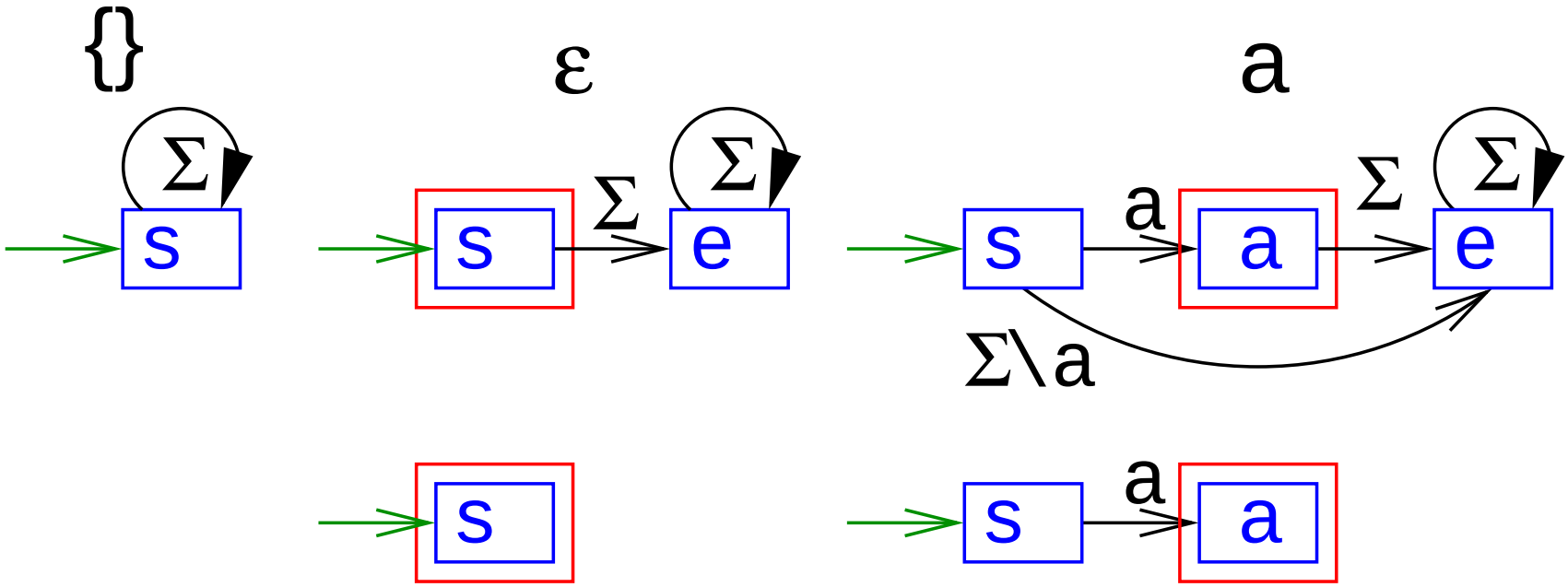
construct the automata for the basic expressions

construct an automaton for complex expression gathering together the automata for the simpler expressions

Theory:

Structural induction on the structure of the regular expressions.

The base case



RegExp→NFA: $L_1 \cup L_2$

$A_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ with $L(A_1) = L_1$

$A_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ with $L(A_2) = L_2$

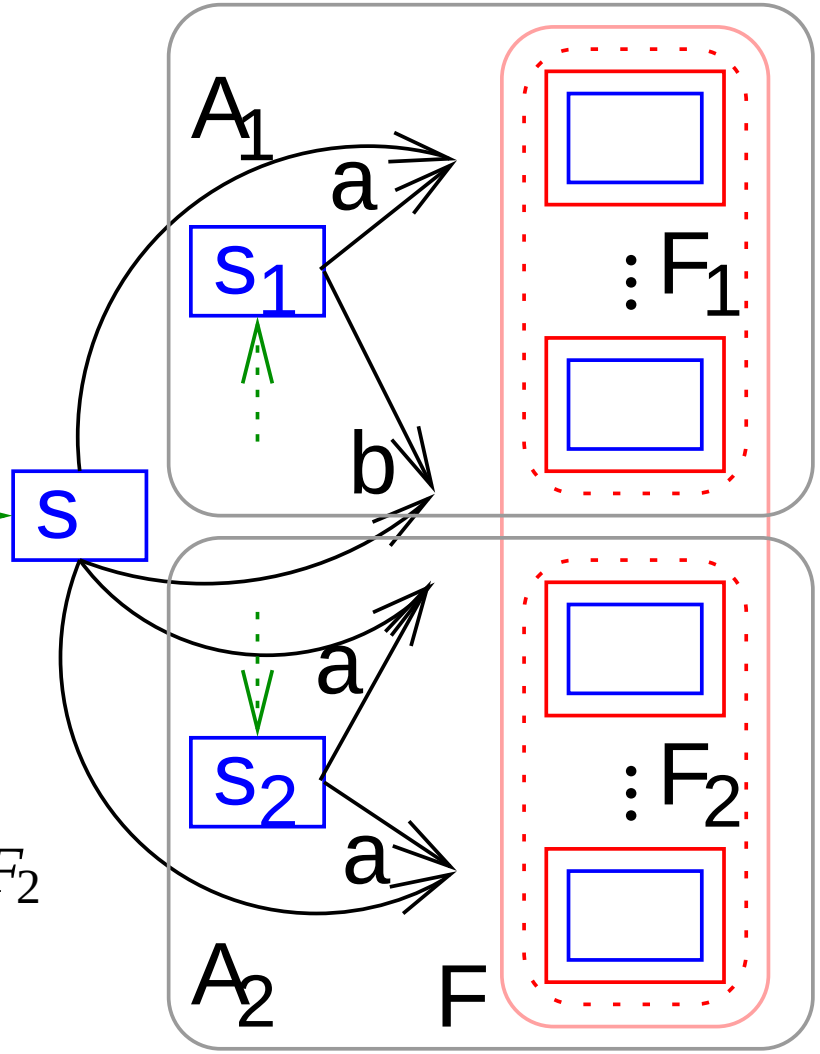
and $Q_1 \cap Q_2 = \emptyset$

$A := (\{s\} \cup Q_1 \cup Q_2, \Sigma, \delta, s, F)$

δ behaves as $\delta_{1/2}$ for $Q_{1/2}$

$\forall a \in \Sigma : \delta(s,a) := \delta(s_1,a) \cup \delta(s_2,a).$

$$F := \begin{cases} F_1 \cup F_2 \cup \{s\} & \text{if } s_1 \in F_1 \vee s_2 \in F_2 \\ F_1 \cup F_2 & \text{otherwise} \end{cases}$$



Proof of $L_1 \cup L_2 \subseteq L(A)$

Let $w \in L_1 = L(A_1)$ (arbitrary).

If $w = \varepsilon$

$\longrightarrow s_1 \in F_1 \longrightarrow s \in F \longrightarrow w \in L(A)$.

If $w = ax$:

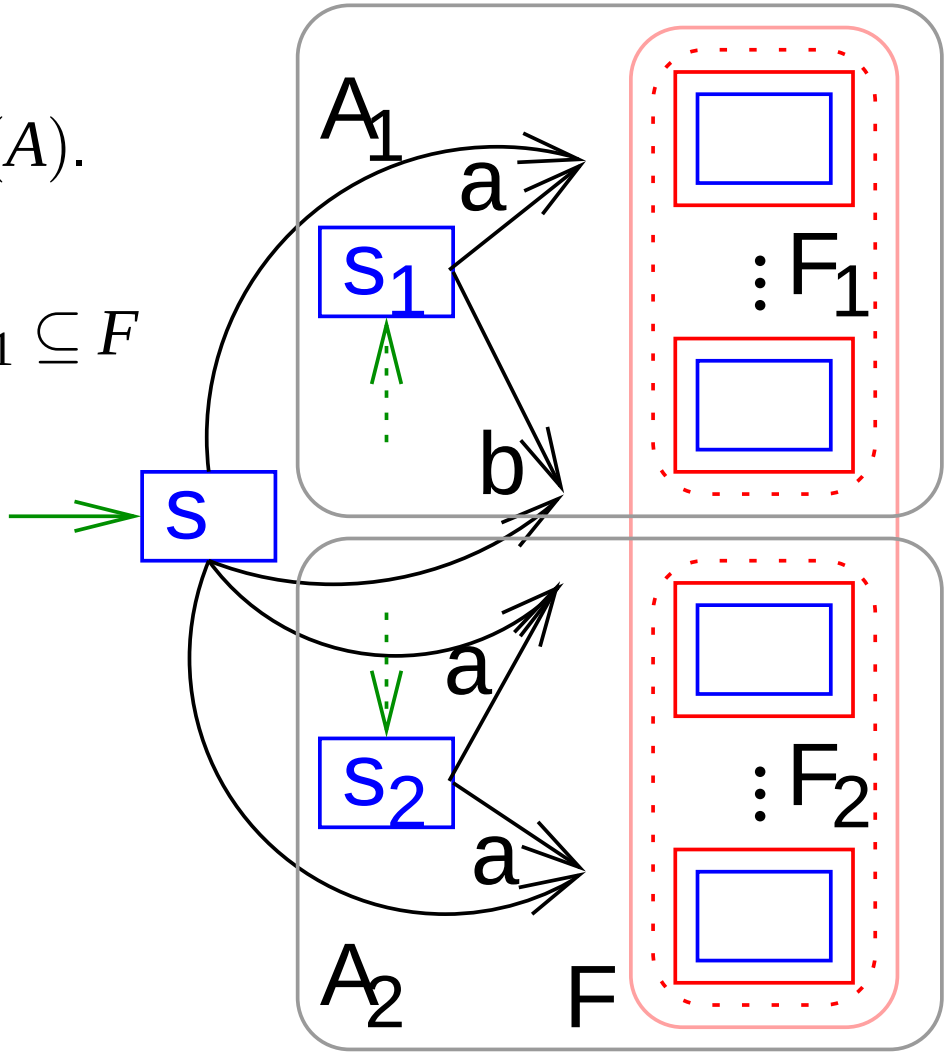
$\longrightarrow \exists$ a path $P_1 = s_1 \overset{a}{\Rightarrow} q_1 \overset{x}{\Rightarrow} f_1 \in F_1 \subseteq F$

$\longrightarrow \exists$ a path $P = s \overset{a}{\Rightarrow} q_1 \overset{x}{\Rightarrow} f_1 \in F$

$\longrightarrow w \in L(A)$.

$w \in L_2 = L(A_2)$

$\longrightarrow \cdots \longrightarrow w \in L(A)$.



Proof of $L(A) \subseteq L_1 \cup L_2$

Let w be an arbitrary $w \in L(A)$.

If $w = \varepsilon \longrightarrow s \in F \longrightarrow s_1 \in F_1 \vee s_2 \in F_2$

$\longrightarrow \varepsilon \in L_1 \vee \varepsilon \in L_2 \longrightarrow \varepsilon \in L_1 \cup L_2$

If $w = ax$:

$\longrightarrow \exists$ a path $P = s \overset{a}{\Rightarrow} q \overset{x}{\Rightarrow} f \in F$.

If $q = q_1 \in Q_1$:

$\longrightarrow \exists$ a path $P_1 = s_1 \overset{a}{\Rightarrow} q_1 \overset{x}{\Rightarrow} f \in F_1$.

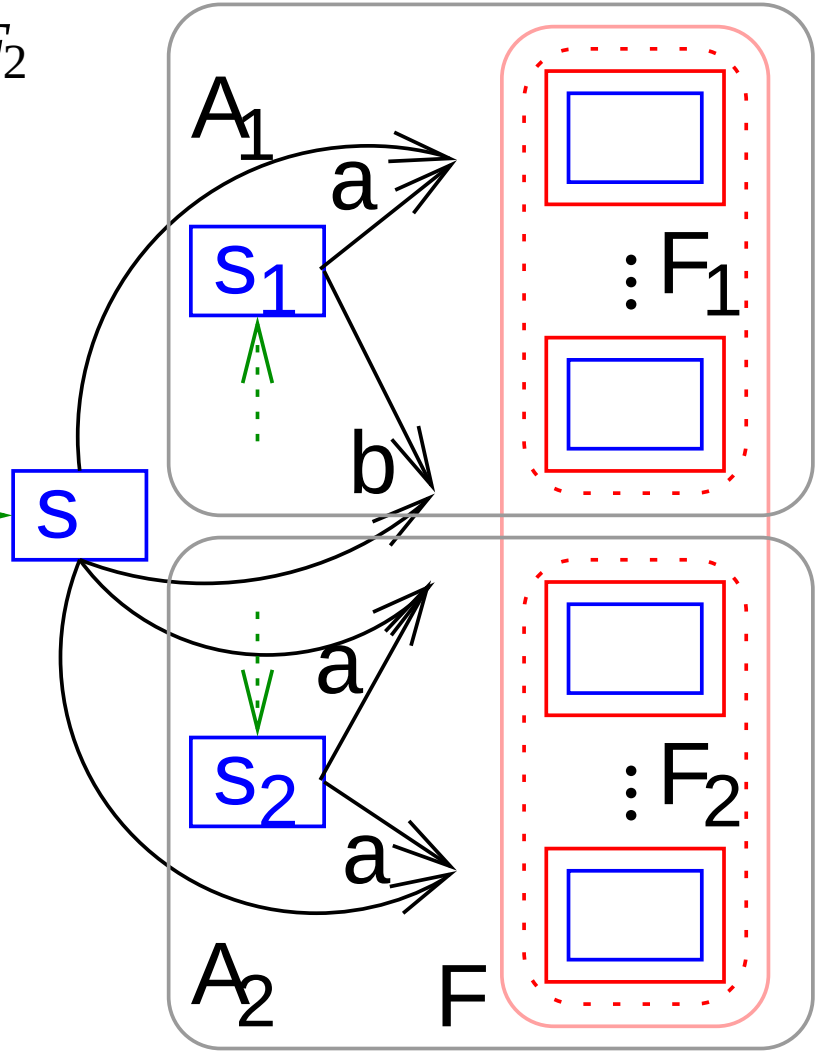
(only states reachable from q_1 are in Q_1.)

$\longrightarrow ax = w \in L_1 \subseteq L_1 \cup L_2$

otherwise: $\longrightarrow q = q_2 \in Q_2$

$\longrightarrow \exists$ a path $P_2 = s_2 \overset{a}{\Rightarrow} q_2 \overset{x}{\Rightarrow} f \in F_2$.

$\longrightarrow ax = w \in L_2 \subseteq L_1 \cup L_2$



□

$L_1 \cdot L_2$

$A_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $L(A_1) = L_1$

$A_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ and $L(A_2) = L_2$

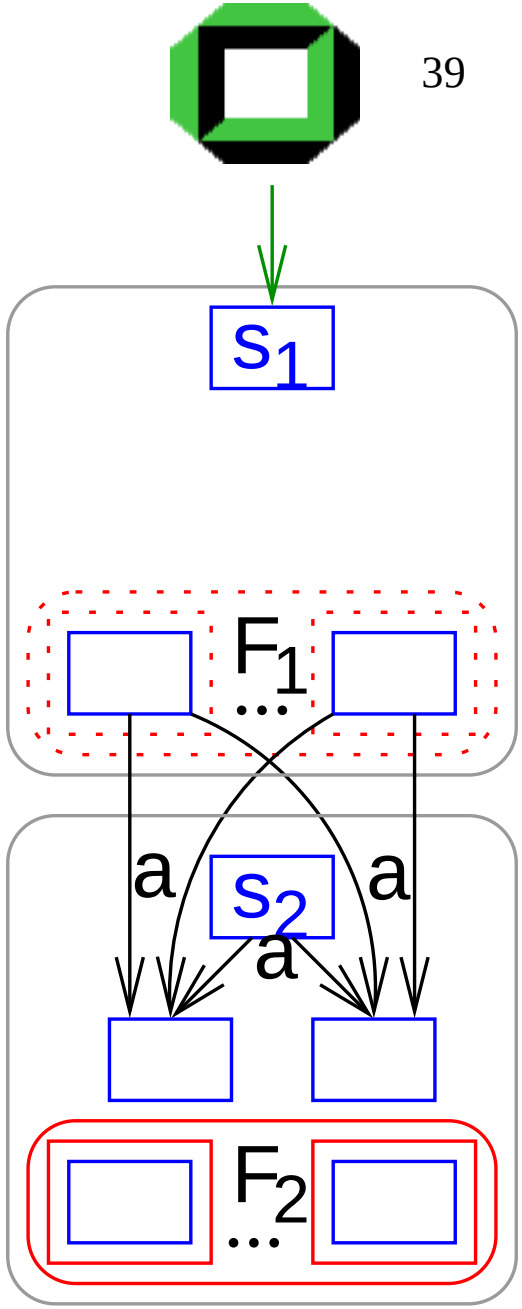
and $Q_1 \cap Q_2 = \emptyset$

$A := (Q_1 \cup Q_2, \Sigma, \delta, s_1, F), \forall a \in \Sigma :$

$$\delta(q,a) := \begin{cases} \delta_1(q,a) & \text{if } q \in Q_1 \setminus F_1 \\ \delta_1(q,a) \cup \delta_2(s_2,a) & \text{if } q \in F_1 \\ \delta_2(q,a) & \text{otherwise} \end{cases}$$

$$F := \begin{cases} F_1 \cup F_2 & \text{if } s_2 \in F_2 \\ F_2 & \text{otherwise} \end{cases}$$

Proof:Exercise

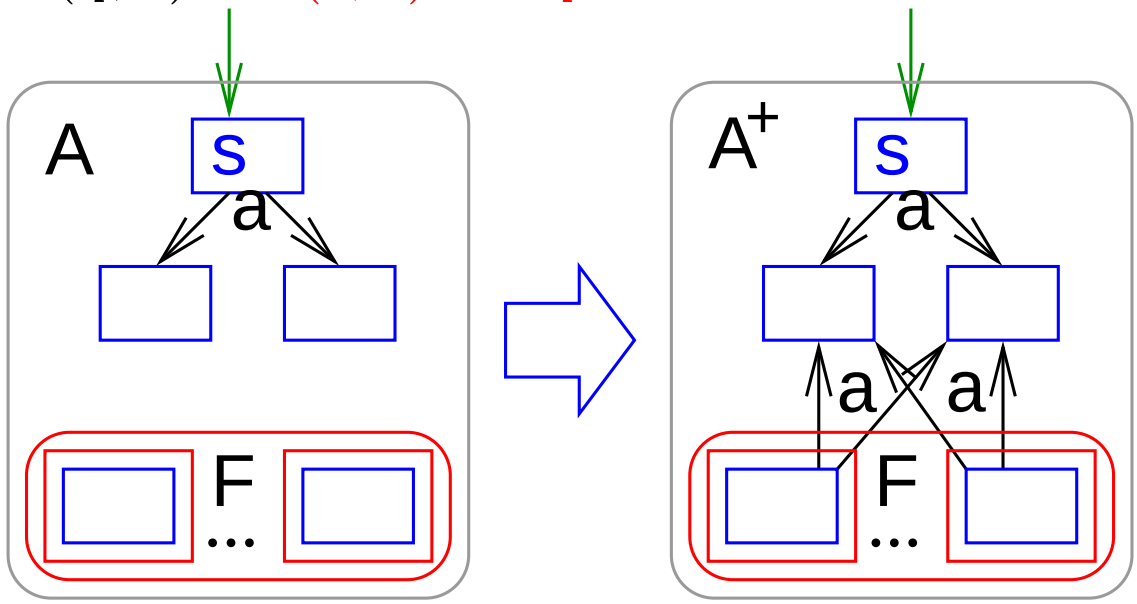


Positive hull $L^+ = \bigcup_{i \geq 1} L^i$

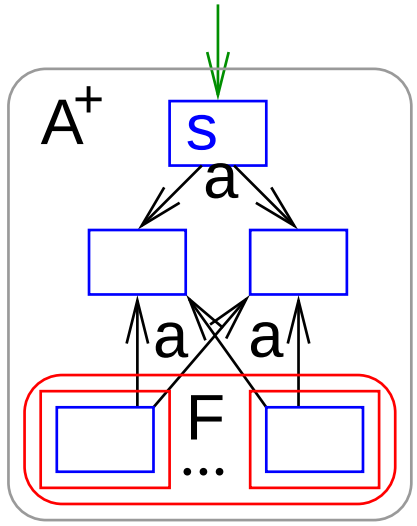
$A = (Q, \Sigma, \delta, s, F)$ and $L(A) = L$

$A^+ := (Q, \Sigma, \delta^+, s, F), \forall a \in \Sigma:$

$$\delta^+(q,a) := \begin{cases} \delta(q,a) & \text{if } q \in Q \setminus F \\ \delta(q,a) \cup \delta(s,a) & \text{if } q \in F \end{cases}$$



Proof of $L(A^+) \subseteq L^+$



Let $w \in L(A^+)$ be an arbitrary and $w \neq \varepsilon$

Let $P = s \overset{a_0}{\Rightarrow} q_0 \overset{*}{\Rightarrow} f$ be an accepting path for w.

Decompose P on transitions of the form $f_j \overset{a_j}{\Rightarrow} q_j$

by $q_j \notin \delta(f_j, a_j)$, $j \in 1..i$, $i \geq 0$.

$\longrightarrow f_j \in F$, $q_j \in \delta(s, a_j)$.

$$P = s \overbrace{\overset{a_0}{\Rightarrow} q_0 \overset{x_0}{\Rightarrow} f_1 \overset{a_1}{\Rightarrow} q_1 \overset{x_1}{\Rightarrow} f_2 \overset{*}{\Rightarrow} f_i \overset{a_i}{\Rightarrow} q_i \overset{x_i}{\Rightarrow}}^{a_0 x_0 a_1 x_1 \cdots a_i x_i = w} f$$

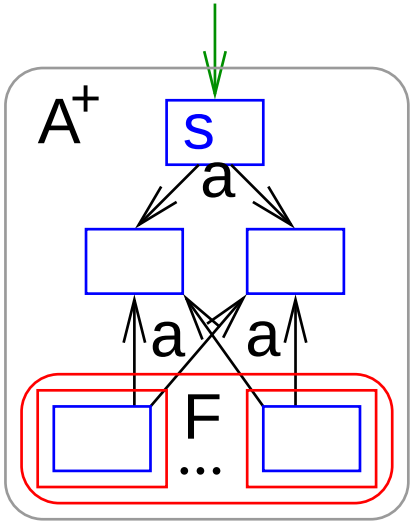
Define $P_j := s \overset{a_j}{\Rightarrow} q_j \overset{x_j}{\Rightarrow} f_{j+1}$ (with $f_{i+1} := f$).

$\longrightarrow \forall j \in 0..i : P_j$ is an accepting path in A.

$\longrightarrow w \in L^+$

□

Proof of $L^i \subseteq L(A^+)$ for $i \geq 1$



Let $w = w_1 \cdots w_i \in L^i$ ($\varepsilon \neq w_i \in L$).

Consider the paths $P_j = s \overset{a_j}{\Rightarrow} q_j \overset{x_j}{\Rightarrow} f_j$, $j \in 1..i$, $f_j \in F$, that testify $w_1 \in L, \ldots, w_i \in L$.

$$\longrightarrow P = s \overbrace{\overset{a_1}{\Rightarrow} q_1 \overset{x_1}{\Rightarrow} f_1 \overset{a_2}{\Rightarrow} q_2 \overset{x_2}{\Rightarrow} f_2 \overset{*}{\Rightarrow} f_{i-1} \overset{a_i}{\Rightarrow} q_i \overset{x_i}{\Rightarrow}}^{w} f_i$$

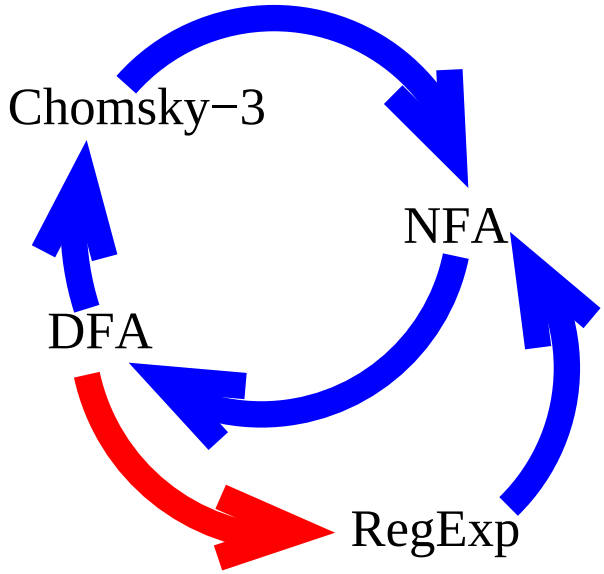
is a path in A^+, testifying that $w \in L(A^+)$.

$\longrightarrow w \in L(A^+)$ □

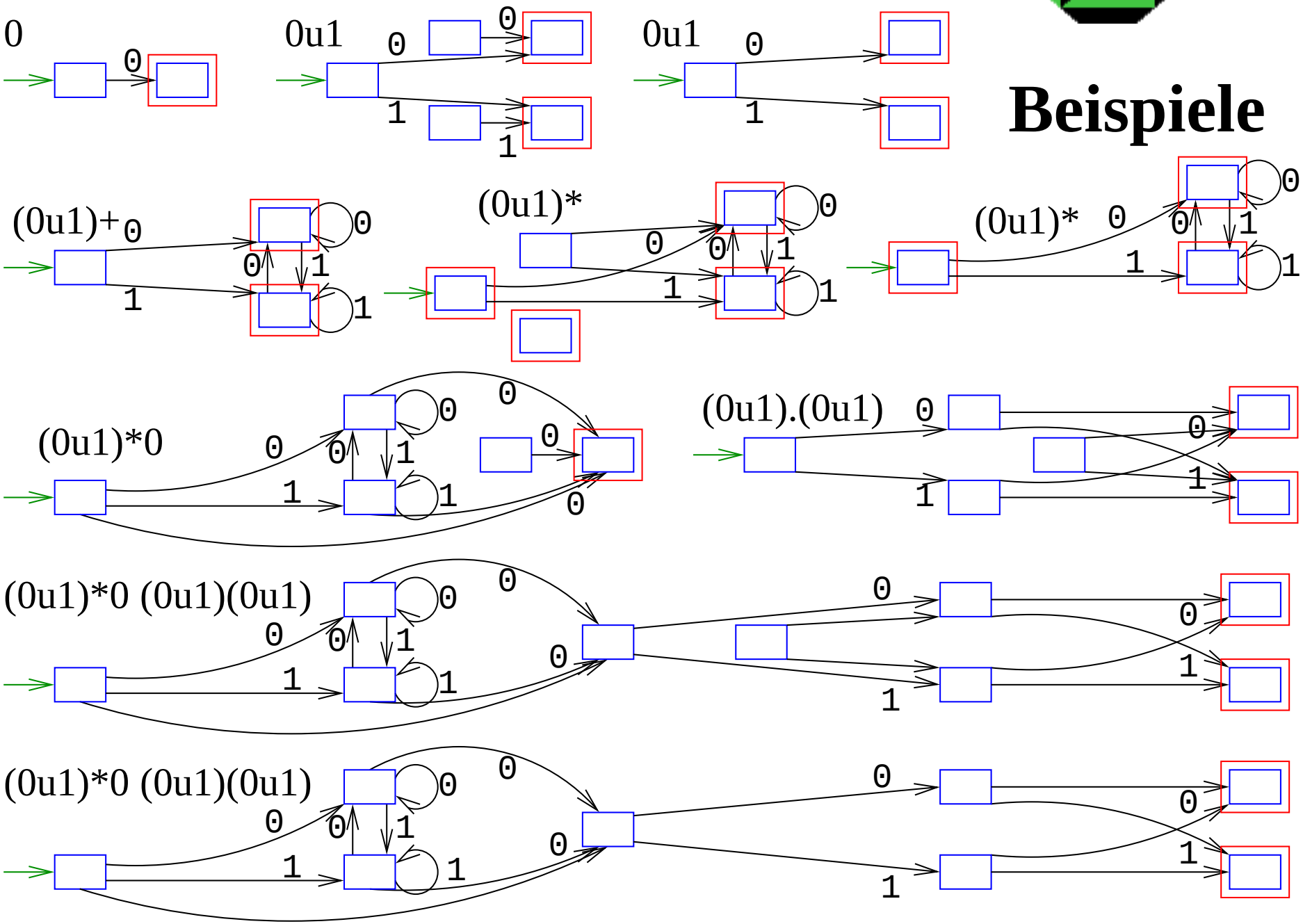
Kleene's hull (star) L^*

Construct an automaton for $\varepsilon \cup L^+ = L^*$.

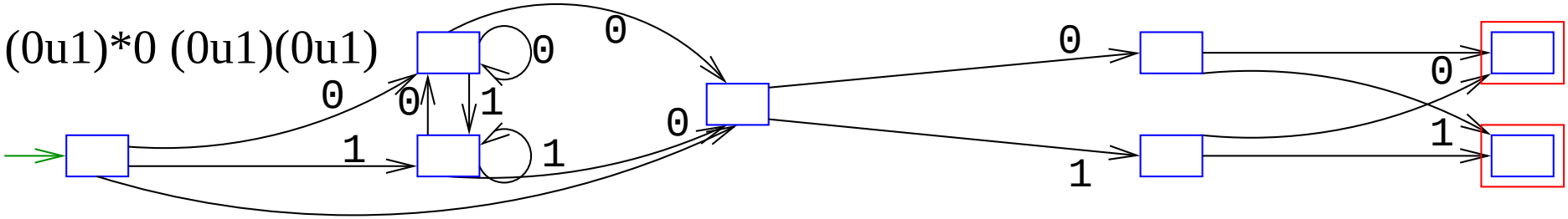
■

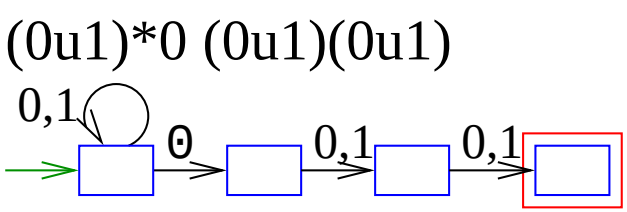


Beispiele



Example





Excursus: Application in searching in the text

Unix-Tool grep: `grep REGULAR-EXPRESSION FILE`

- Search of all substrings in FILE, which are in $L(\text{REGULAR-EXPRESSION})$
- A lot of syntax: Scope `a-g`, expression for example `:alnum:`,default number of iterations,...
- But it is very easy if we translate this in a regular expression.
- Fast implementation by translation in a finite deterministic automaton.

Application in scanner-(generator), `lex`, `flex`

Input: A system for regular expressions

Output: A finite automaton (C code),

Runtime-input: The program as strings

Runtime-output: The program for token (packets) like numbers, identifiers, keywords.

- time-critical, since only here each particular symbol will be touched, comments will be dropped, decimal numbers transformed to binary ,. . .
- it makes the presentment standard by removing the blanks from the number presentation ,. . .
- it simplifies the followed syntax analyze

Similar functionality

- Editors, for example emacs
- Script-languages like Perl (notorious?)
- `java.util.regex` Library
- C++ Boost.Regex Library
- .net framework
- Preprocessing by parsing for xml documents

Excursus: ε-transitions

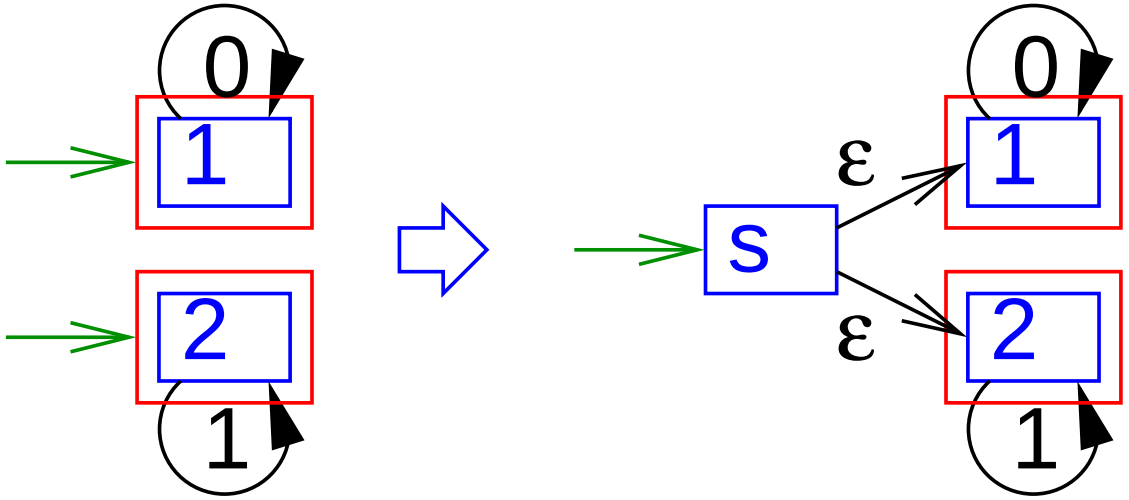
We allow spontaneous transition

without consumption an input symbol.

εNFA

- Q, the set of states
- Σ, the alphabet
- $\delta : Q \times (\Sigma \cup \varepsilon) \rightarrow 2^Q$, the transition function
- $s \in Q$, initial state
- $F \subseteq Q$, final states

Examples: $0^* \cup 1^*$



RegExp$\rightarrow$ εNEA: $L_1 \cup L_2$

$A_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ with $L(A_1) = L_1$

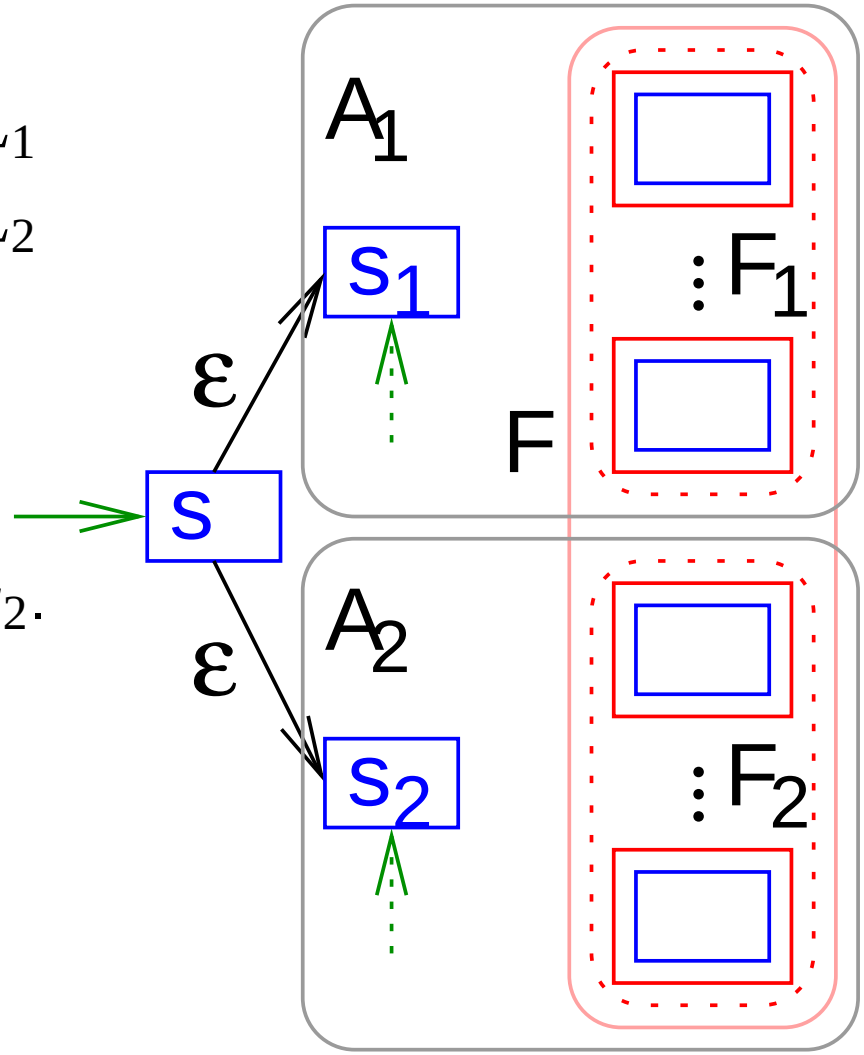
$A_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ with $L(A_2) = L_2$

and $Q_1 \cap Q_2 = \emptyset$

A:= $(\{s\} \cup Q_1 \cup Q_2, \Sigma, \delta, s, F_1 \cup F_2)$

δ behaves as $\delta_{1/2}$ on $Q_{1/2}$

spontaneous transition from s to s_1 and s_2.



$L_1 \cdot L_2$

$A_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ where $L(A_1) = L_1$

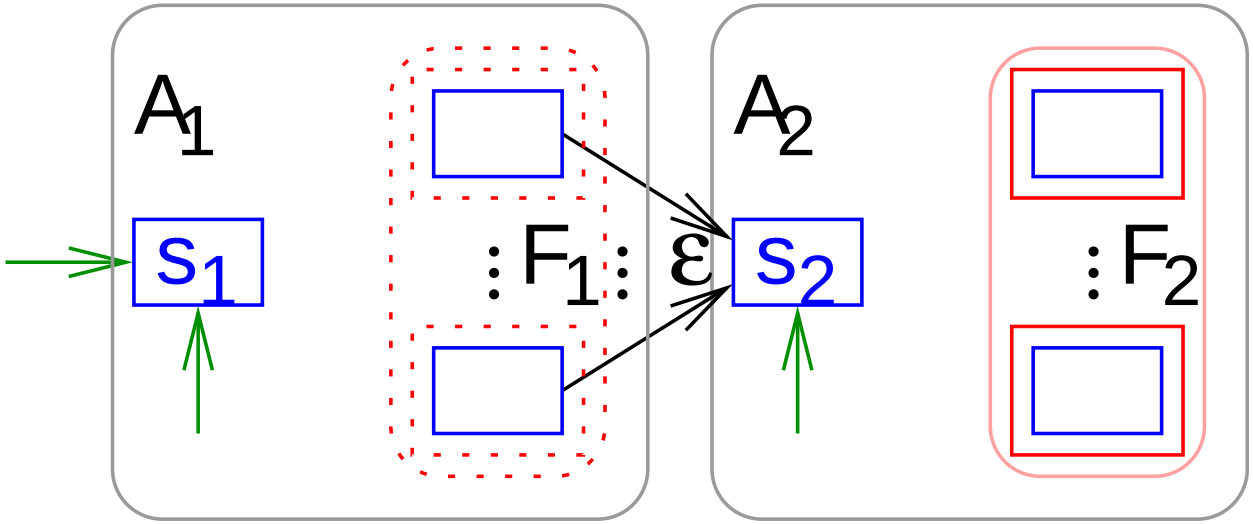
$A_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ where $L(A_2) = L_2$

and $Q_1 \cap Q_2 = \emptyset$

$A := (Q_1 \cup Q_2, \Sigma, \delta, s_1, F_2)$

δ behaves as $\delta_{1/2}$ on $Q_{1/2}$

spontaneous transition from F_1 to s_2.



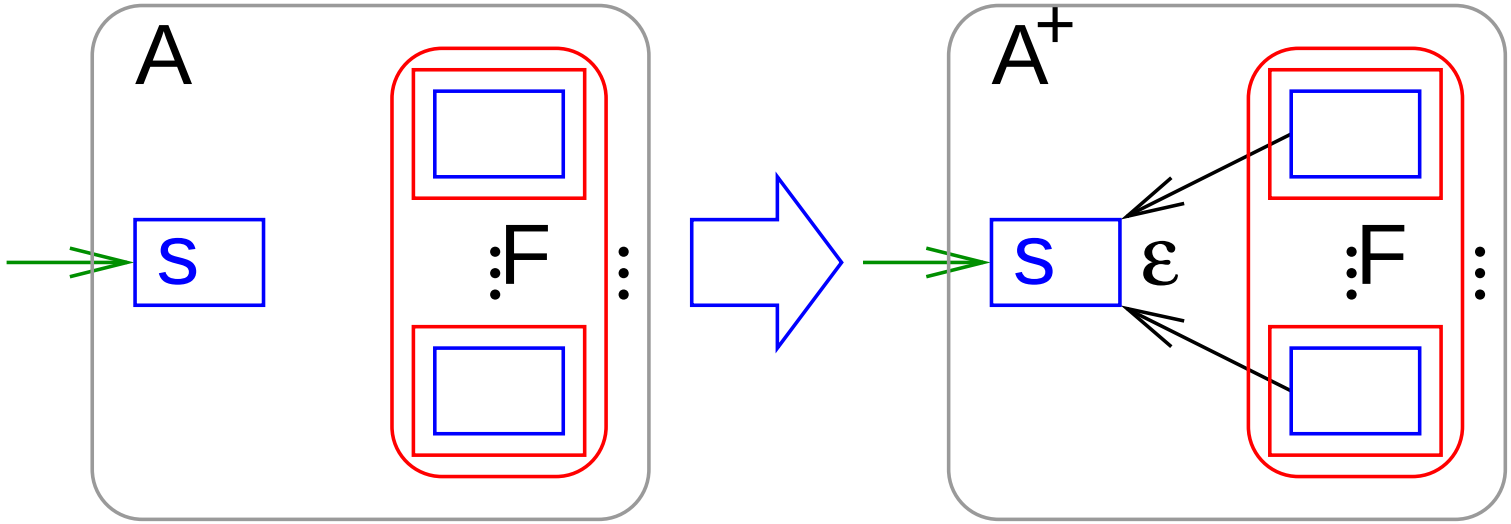
Positive hull $L^+ = \bigcup_{i \geq 1} L^i$

$A = (Q, \Sigma, \delta, s, F)$ with $L(A) = L$

$A^+ := (Q, \Sigma, \delta^+, s, F)$

δ^+ behaves as δ

spontaneous transition $f \rightarrow s \forall f \in F$.



ε**NFA** $A \rightsquigarrow$**NFA** $\overline{A}$

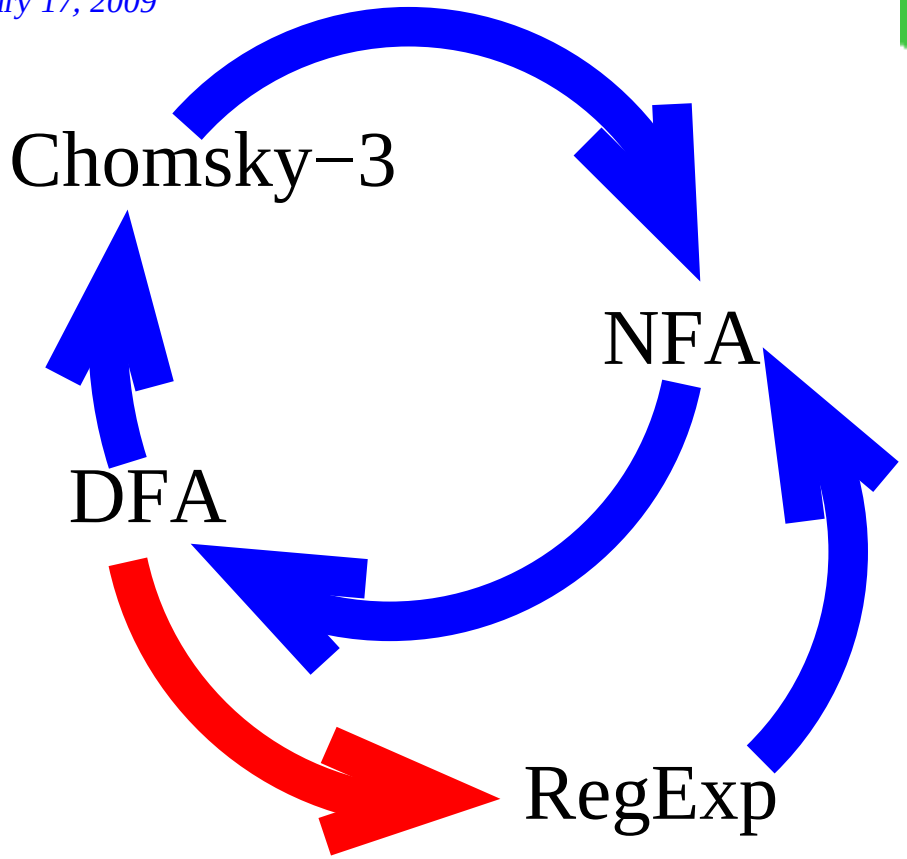
Proof (idea):

replace in each path with ε and one letter in A by explicit transitions in $\overline{A}$.

Be careful with the final states.

DFA→RegExp

Goal:



- regular expressions are universal interface
- reverse engineering (maybe)

Proof: Given: DFA $A = (\{1,\ldots,n\},\Sigma,\delta,s,F)$

Result: regular expression α with $L(A) = L(\alpha)$.

For $f \in F$ let $L_f = \left\{ w \in \Sigma^* : \hat{\delta}(s,w) = f \right\}$.

We will find a RegExp for L_f. Since $L(A) = \bigcup_{f \in F} L_f$, we are done.

Given: DFA $A_f = (\{1,\ldots,n\},\Sigma,\delta,s,\{f\})$

Result: regular expression α with $L_f = L(A_f) = L(\alpha)$.

Let $L_{ij} := L((\{1,\ldots,n\},\Sigma,\delta,i,\{j\}))$

In particular $L_{sf} = L_f$.

$L_{ij}^m := \left\{ w \in \Sigma^* : \exists \text{working path } i \overset{w}{\Rightarrow} j = iPj \text{ with } P \in \{1,\ldots,m\}^* \right\}$

Notice that $L_{ij} = L_{ij}^n$.

We construct a regular expression for L_{ij}^m inductively

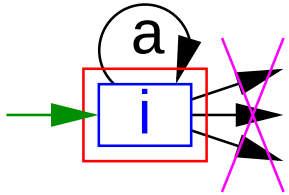
using the regular expressions for the smaller m.

$$L_{ij}^m := \left\{ w \in \Sigma^* : \exists \text{working path } i \overset{w}{\Rightarrow} j = iPj \text{ with } P \in \{1,\ldots,m\}^* \right\}$$

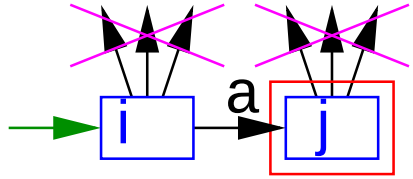
Given: regular expression α_{ij}^k, $k < m$ with $L(\alpha_{ij}^k) = L_{ij}^k$

Result: α_{ij}^m with $L(\alpha_{ij}^m) = L_{ij}^m$

If $m = 0, i = j$**:** $\alpha_{ii}^0 = \bigcup_{a \in \Sigma : \delta(i,a) = i} a \cup \varepsilon$

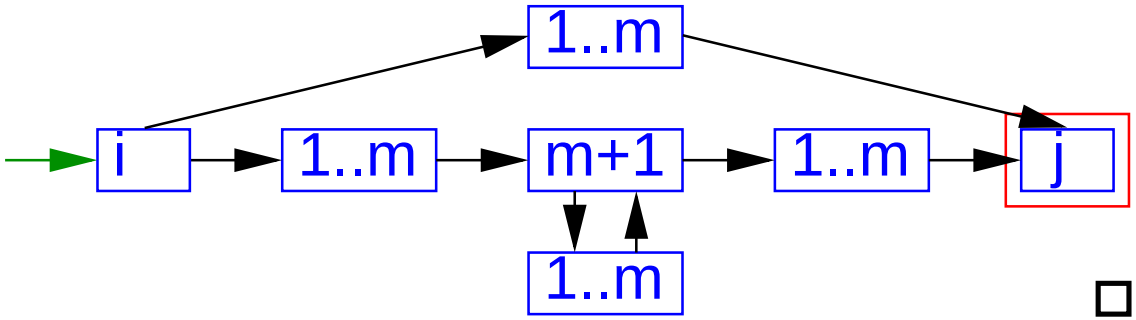


If $m = 0, i \neq j$**:** $\alpha_{ij}^0 = \bigcup_{a \in \Sigma : \delta(i,a) = j} a$



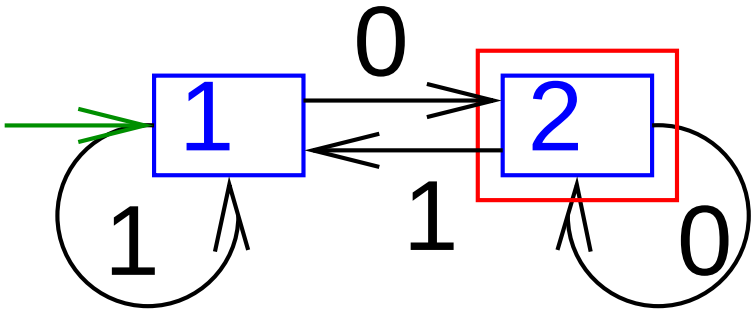
If $m \rightsquigarrow m+1$**:**

$$\alpha_{ij}^{m+1} = \alpha_{ij}^m \cup \alpha_{i,m+1}^m \cdot (\alpha_{m+1,m+1}^m)^* \cdot \alpha_{m+1,j}^m$$



□

Example



$$\alpha_{11}^0 = 1 \cup \varepsilon \qquad \alpha_{22}^0 = 0 \cup \varepsilon \qquad \alpha_{12}^0 = 0 \qquad \alpha_{21}^0 = 1$$

$$\begin{aligned} \alpha_{12}^1 &= \alpha_{12}^0 \cup \alpha_{11}^0 \cdot (\alpha_{11}^0)^* \cdot \alpha_{12}^0 \\ &= 0 \cup (1 \cup \varepsilon) \cdot (1 \cup \varepsilon)^* \cdot 0 \\ &= 1^* 0 \end{aligned}$$

$$\begin{aligned} \alpha_{22}^1 &= \alpha_{22}^0 \cup \alpha_{21}^0 \cdot (\alpha_{11}^0)^* \\ &= 0 \cup \varepsilon \cup 1 \cdot (1 \cup \varepsilon)^* \\ &= 1^* 0 \cup \varepsilon \end{aligned}$$

$$\begin{aligned} \alpha_{12}^2 &= \alpha_{12}^1 \cup \alpha_{12}^1 \cdot (\alpha_{22}^1)^* \cdot \alpha_{22}^1 \\ &= 1^*0 \cup 1^*0 \cdot (1^*0 \cup \varepsilon)^* \cdot (1^*0 \cup \varepsilon) \\ &= 1^*0(1^*0)^* \end{aligned}$$

$L(\alpha_{12}^2) = L_{12}^2 = L_{12} = L_2$, where $F = \{2\}$.

1.2.4 The Pumping Lemma (for regular languages)

L regular

$$\longrightarrow \exists n \in \mathbb{N} : \forall w \in L : |w| > n$$

$$\longrightarrow \exists u, v, x : w = uvx \wedge$$

1. $|v| \geq 1 \wedge$
2. $|uv| \leq n \wedge$
3. $\forall i \in \mathbb{N}_0 : uv^i x \in L$

In words:

A sufficiently long word of a regular language has a non trivial middle part which by iteration we could „ pump“.

Proof of the Pumping Lemma

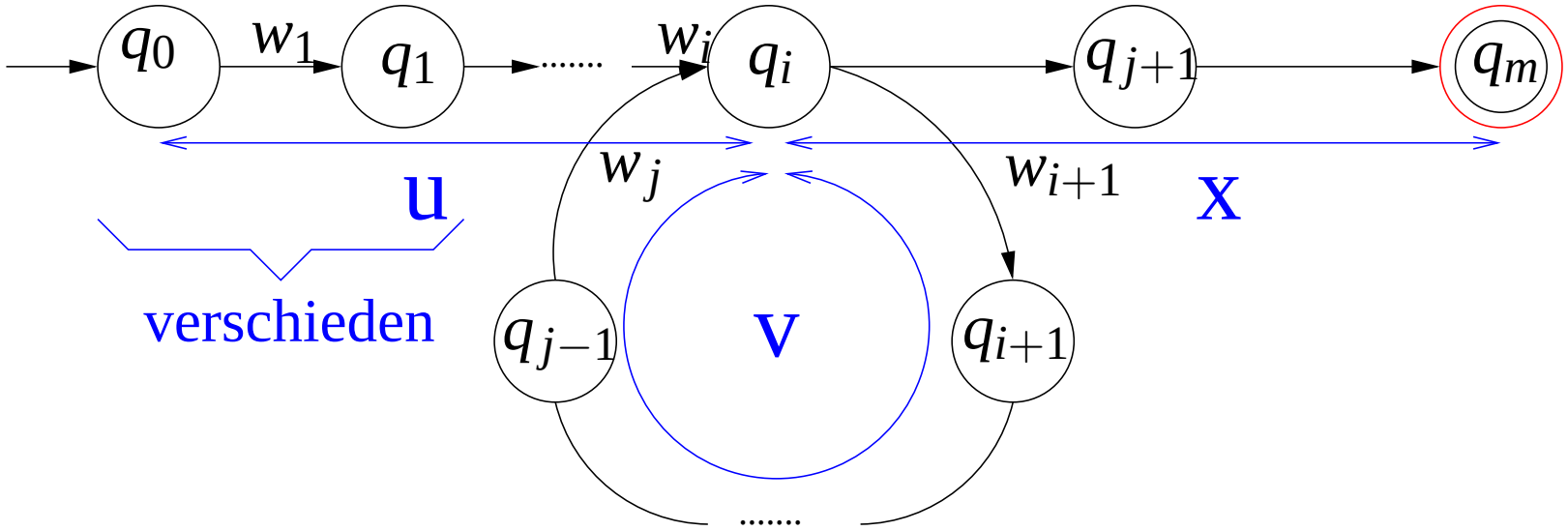
L regular $\longrightarrow \exists n \in \mathbb{N} : \forall w \in L : |w| > n \longrightarrow \exists u, v, x :$

$w = uvx \wedge |v| \geq 1 \wedge |uv| \leq n \wedge \forall i \in \mathbb{N}_0 : uv^i x \in L$

Proof: Let $A = (Q, \Sigma, \delta, q_0, F)$ DFA with $L(A) = L$.

Let $n = |Q|$ and $w \in L$ with $|w| = m > n$ (arbitrary).

Let $q_0, \ldots, q_m$ be the traversed states.



$(\exists i < j \leq n : q_i = q_j) \longrightarrow |v| \geq 1, |uv| \leq n, uv^i x$ will be accepted as well.

Example: $L = \{a^k b^k : k \in \mathbb{N}\}$

Suppose that L is regular.

Let n be the number from the Pumping Lemma and consider

$w = a^n b^n = uvx$ in accordance of the Pumping Lemma, then $ux \in L$.

$|uv| \leq n, |v| \geq 1 \longrightarrow v = a^\ell$ for a $\ell \geq 1$.

$ux = a^{n-\ell} b^n \in L$.

A contradiction. ■

Example: Balanced parentheses $L_{()}$

Assume that L is regular.

Let n be the number from the Pumping Lemma and consider

$w = (^n)^n = uvx$ according to the Pumping Lemma

$ux \in L_{()}$ and $|v| > 1$ and $|uv| \leq n$.

Then $v = (^i$

and $ux = (^{n-i})^n \notin L_{()}$ A contradiction.

$L = \{0^p : p$ **is a prime number**$\}$

Assume that L is regular.

Let n be the number from the Pumping Lemma.

Let $p \geq n+2$ be a prime number. ($\exists$ infinitely many prime numbers)

$\longrightarrow 0^p \in L = uvw, |v| \geq 1, |uw| \geq 2.$

Pumping-Lemma: $uv^{|uw|}w \in L$.

$\longrightarrow |uw| + |uw| \cdot |v| = |uw|(1+|v|)$ is a prime number.

Two nontrivial factors $|uw| \geq 2$ and $(1+|v|) \geq 2$.

A contradiction. ■

The Pumping-Lemma is

not enough condition for regularity

Example: $L = \{c^m a^\ell b^\ell : m, \ell \geq 0\} \cup \{a, b\}^*$ is not regular, but

let $n \geq 1$, $x \in L$ arbitrary with $|x| \geq n$.

if $x \in a^* b^*$**:**

Write $x = \underbrace{\varepsilon}_{u} \underbrace{a}_{v} \underbrace{a^m b^{n-m-1}}_{w}$

1. $|v| = 1 \geq 1$
2. $|uv| = 1 \leq n$
3. $uv^i w = a^i a^m b^{n-m-1} \in a^* b^* \subseteq L$

The Pumping-Lemma is

not enough condition for regularity

Example: $L = \{c^m a^\ell b^\ell : m, \ell \geq 0\} \cup \{a,b\}^*$ is not regular,

Let n (actually) arbitrary, $w \in L$ arbitrary with $|w| \geq n$.

If $w \in a^*b^*$**:** No Problem.

If $w = c^m a^\ell b^\ell$**,** $m \geq 1$**:**

Write $w = \underbrace{\varepsilon}_{u}\underbrace{c}_{v}\underbrace{c^{m-1}a^\ell b^\ell}_{x}$

1. $|v| = 1 \geq 1$
2. $|uv| = 1 \leq n$
3. $uv^i x = c^{m-1+i}a^\ell b^\ell \in L$

1.2.5 Equivalence relations and Minimal automaton

Idea: work directly with L without a reference to a concrete automaton.

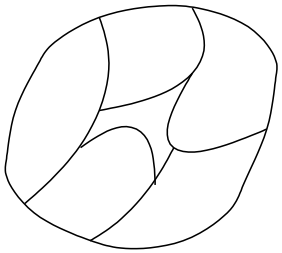
Reminder: Equivalence relation

A relation $R \subseteq Y \times Y$ is called equivalence relation if R is:

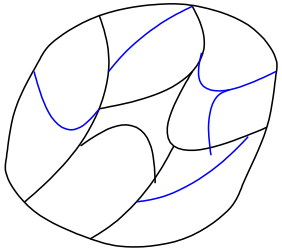
- ☐ reflexive $\forall x : xRx$
- ☐ transitive $\forall xyz : xRy \wedge yRz \longrightarrow xRz$
- ☐ symmetric. $\forall xy : xRy \longrightarrow yRx$

Equivalence class: $[x] = \{y : xRy\}$. eq.-classes are nonempty and pairwise disjoint, i.e.

every element of Y belongs to exactly one eq.-class.



Index: index(R):= $|\text{Equivalence classes}| = |\{[x] : x \in Y\}|$



Refinement: R **refines** R' **(**$R \subseteq R'$**)**

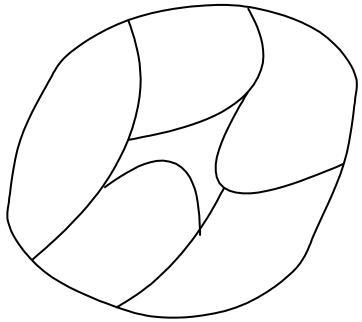
Lemma: R refines $R' \longrightarrow \forall$ equivalence class $[x]_R : [x]_R \subseteq [x]_{R'}$

Proof:

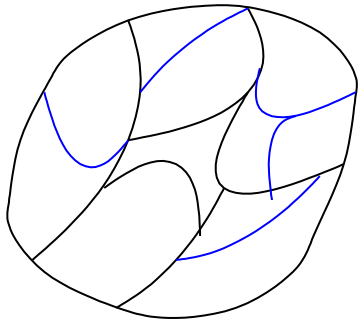
$$y \in [x]_R \Leftrightarrow (y,x) \in R$$

$$\xrightarrow{R \subseteq R'} (y,x) \in R'$$

$$\Leftrightarrow y \in [x]_{R'}$$



Corollary: R refines $R' \longrightarrow \text{index}(R) \geq \text{index}(R')$



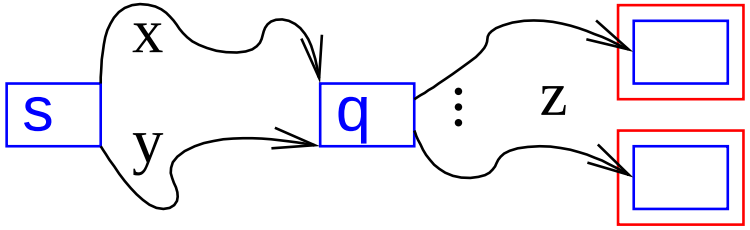
Nerode Relation

The Nerode Relation for the language L is defined as

$$R_L := \{(x,y) \in \Sigma^* \times \Sigma^* : \forall z \in \Sigma^* : xz \in L \Leftrightarrow yz \in L\}$$

Idea: the equivalence classes correspond to the states.

Why? Particulary!



DFAs induce equivalence relations

Let $M = (Q, \Sigma, \delta, s, F)$ DFA with $L(M) = L$.

$R_M := \left\{(x,y) \in \Sigma^* \times \Sigma^* : \hat{\delta}(s,x) = \hat{\delta}(s,y)\right\}$.

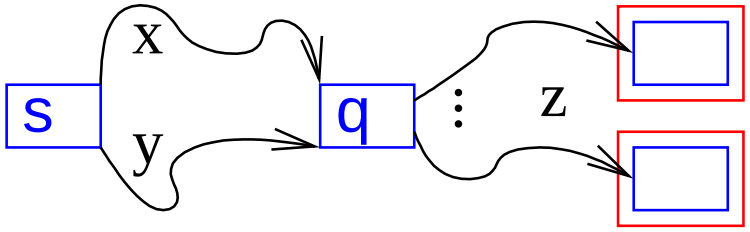
Equivalence relation! An equivalence class per state (reachable from s).

Lemma 1: R_M refines the Nerode relation $R_L =$

$$\{(x,y) \in \Sigma^* \times \Sigma^* : \forall z \in \Sigma^* : xz \in L \Leftrightarrow yz \in L\}$$

Proof (sketch):

$$\forall (x,y) \in \Sigma^* \times \Sigma^* : \hat{\delta}(s,x) = \hat{\delta}(s,y) \longrightarrow \forall z : xz \in L \Leftrightarrow yz \in L$$



Infinite index of the Nerode relation

Observation: index$(R_L) = \infty \longrightarrow L$ is not regular.

Proof: Assume that L is regular.

$\longrightarrow \exists$ DFA $M = (Q, \Sigma, \delta, s, F) : L(M) = L$.

$\longrightarrow R_M$ refines R_L.

$\longrightarrow |Q| \geq \text{index}(R_M) \geq \text{index}(R_L) = \infty$.

A contradiction

Hence: if L is regular, then index$(R_L) < \infty$.

Equivalence-classes automaton

$R_L := \{(x,y) \in \Sigma^* \times \Sigma^* : \forall z \in \Sigma^* : xz \in L \Leftrightarrow yz \in L\}$

Idea: when the equivalence classes $[w_1], \ldots, [w_k]$ of R_L correspond to the states of a DFA $M_\equiv$, then by Lemma below the state-minimal automaton for L is:

$M_\equiv := (\{[w_1], \ldots, [w_k]\}, \Sigma, \delta_\equiv, [\varepsilon], F_\equiv)$ with
$F_\equiv := \{[w] : w \in L\}$ and
$\delta_\equiv([w], a) := [wa]$.

Lemma: $\delta_\equiv$ is well defined
Lemma: $\hat{\delta}_\equiv([\varepsilon], w) = [w]$
Lemma: $L(M_\equiv) = L$

Equivalence-classes automaton

$R_L := \{(x,y) \in \Sigma^* \times \Sigma^* : \forall z \in \Sigma^* : xz \in L \Leftrightarrow yz \in L\}$

$M_\equiv := (\{[w_1], \ldots, [w_k]\}, \Sigma, \delta_\equiv, [\varepsilon], F_\equiv)$ where

$F_\equiv := \{[w] : w \in L\}$ and

$\delta_\equiv([w], a) := [wa]$.

Lemma: $\delta_\equiv$ is well defined

So $xR_Ly \longrightarrow \forall a \in \Sigma : xaR_Lya$ *right-invariance*

$xR_Ly \longrightarrow \forall z \in \Sigma^* : xz \in L \Leftrightarrow yz \in L$ especially

$\longrightarrow \forall az \in \Sigma^* : x(az) \in L \Leftrightarrow y(az) \in L$

$\Leftrightarrow \forall a \in \Sigma : \forall z \in \Sigma^* : (xa)z \in L \Leftrightarrow (ya)z \in L$

$\longrightarrow \forall a \in \Sigma : xaR_Lya$ □

Equivalence-classes automaton

$R_L := \{(x,y) \in \Sigma^* \times \Sigma^* : \forall z \in \Sigma^* : xz \in L \Leftrightarrow yz \in L\}$

$M_\equiv := (\{[w_1], \ldots, [w_k]\}, \Sigma, \delta_\equiv, [\varepsilon], F_\equiv)$ where

$F_\equiv := \{[w] : w \in L\}$ and

$\delta_\equiv([w], a) := [wa]$.

Lemma: $\hat{\delta}_\equiv([x], y) = [xy]$

Induction on $|y|$:

$\hat{\delta}_\equiv([x], \varepsilon) = [x]$.

$\hat{\delta}_\equiv([x], aw) \overset{\text{Def.}\hat{\delta}_\equiv}{=} \hat{\delta}_\equiv(\delta_\equiv([x], a), w) \overset{\text{Def.}\delta_\equiv}{=} \hat{\delta}_\equiv([xa], w) \overset{IH}{=}$

$[xaw]$. □

Equivalence-classes automaton: accepts L

$R_L := \{(x,y) \in \Sigma^* \times \Sigma^* : \forall z \in \Sigma^* : xz \in L \Leftrightarrow yz \in L\}$

$M_\equiv := (\{[w_1], \ldots, [w_k]\}, \Sigma, \delta_\equiv, [\varepsilon], F_\equiv)$ with

$F_\equiv := \{[w] : w \in L\}$ and

$\delta_\equiv([w], a) := [wa]$.

Lemma: $L(M_\equiv) = L$.

$w \in L(M_\equiv)$

$\Leftrightarrow \hat{\delta}_\equiv([\varepsilon], w) \in \{[w] : w \in L\}$ Def. $M_\equiv$

$\Leftrightarrow [w] \in \{[w] : w \in L\}$ Previous Lemma

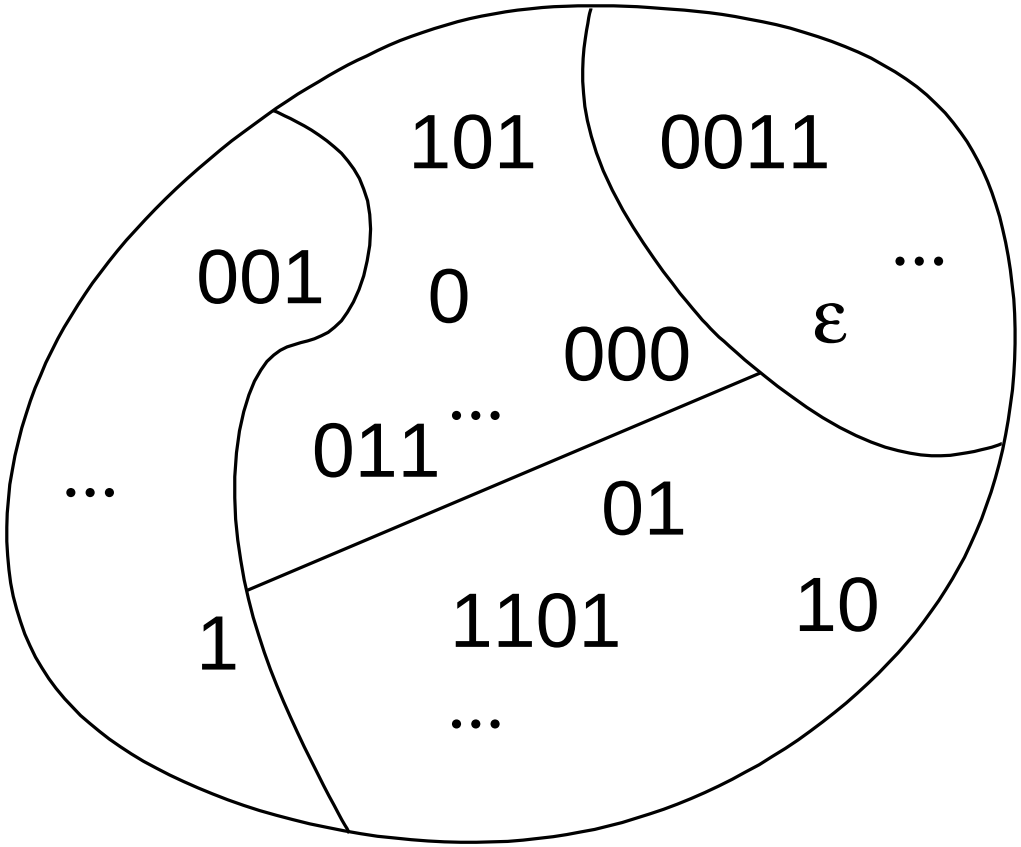
$\Leftrightarrow w \in L$. Eq. classes are entire or not at all in L

$([w] \in \{[w] : w \in L\} \longrightarrow \exists x \in L : [x] = [w] \longrightarrow xR_Ly \longrightarrow \forall z :$

$xz \in L \Leftrightarrow wz \in L \longrightarrow x\varepsilon \in L \Leftrightarrow w\varepsilon \in L)$

Example

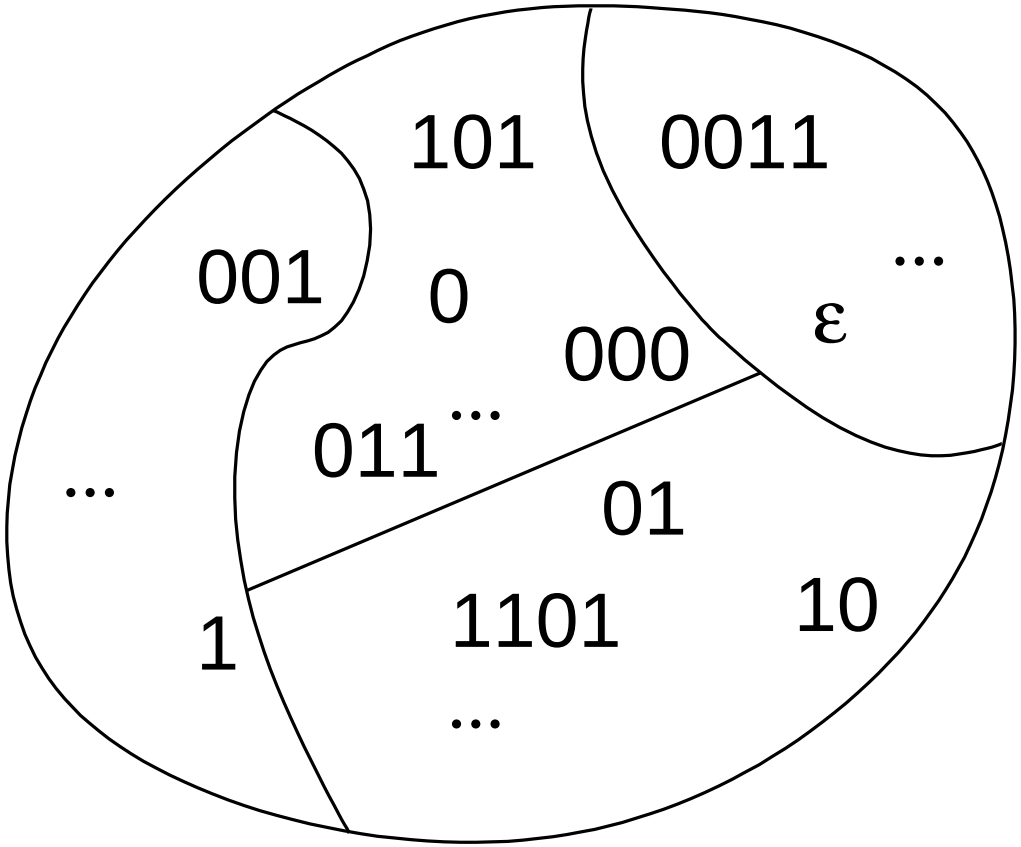
$L \subseteq \{0,1\}^*$ language, all words with even number of one's and even number of zero's



Equivalence-classes?

Example

$L \subseteq \{0,1\}^*$ language, all words with even number of one's and even number of zero's

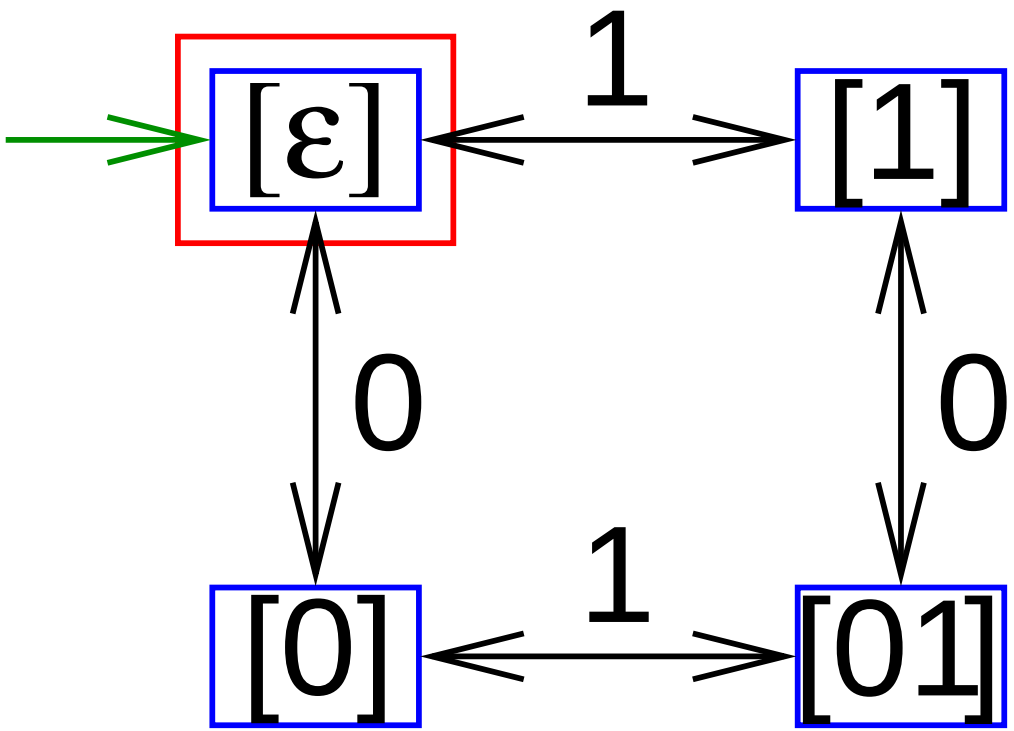


Equivalence-classes:

$[\varepsilon], [0], [1], [01]$

Example

$L \subseteq \{0,1\}^*$ language, all words with even number of one's and even number of zero's



Theorem of Nerode

Let

$R_L := \{(x,y) \in \Sigma^* \times \Sigma^* : \forall z \in \Sigma^* : xz \in L \Leftrightarrow yz \in L\}$.

L not regular $\longrightarrow \text{index}(R_L) = \infty$

L regular $\longrightarrow$ Let

$M_{\equiv} := (\{[w_1], \ldots, [w_k]\}, \Sigma, \delta_{\equiv}, [\varepsilon], F_{\equiv})$, and

$M = (Q, \Sigma, \delta, s, F)$ arbitrary DFA with $L(M) = L$.

It holds $L(M_{\equiv}) = L$ and R_M is refinement for R_L.

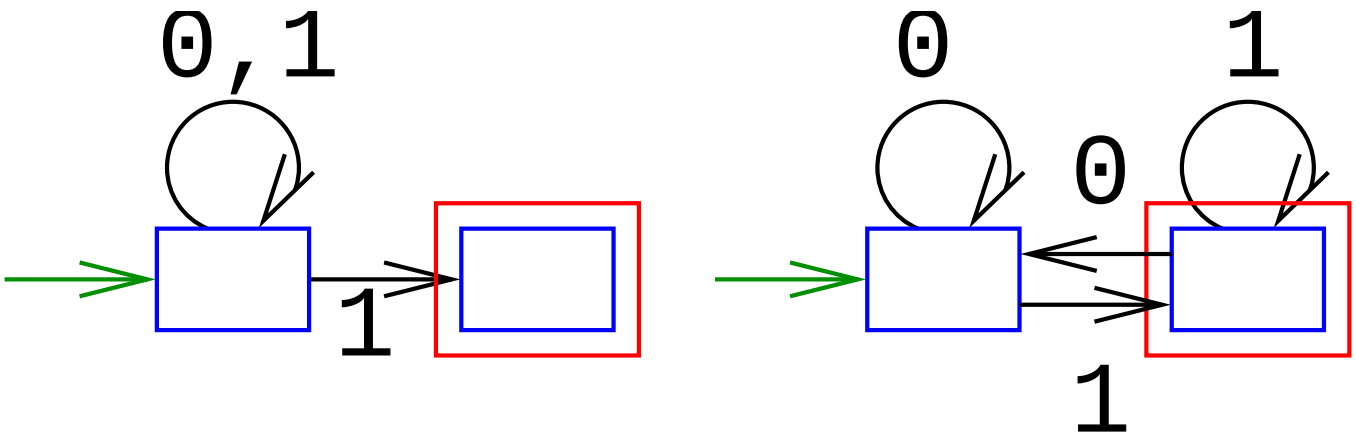
Corollary:

All state-minimal automata for L are isomorphic to $M_{\equiv}$.

(equal except for renaming)

A counterexample for NFA

There are structural different state-minimal NFAs for $(0 \cup 1)^* 1$.



Exercise: Write down the transition functions.

Construction

of the state-minimal automaton

Preprocessing

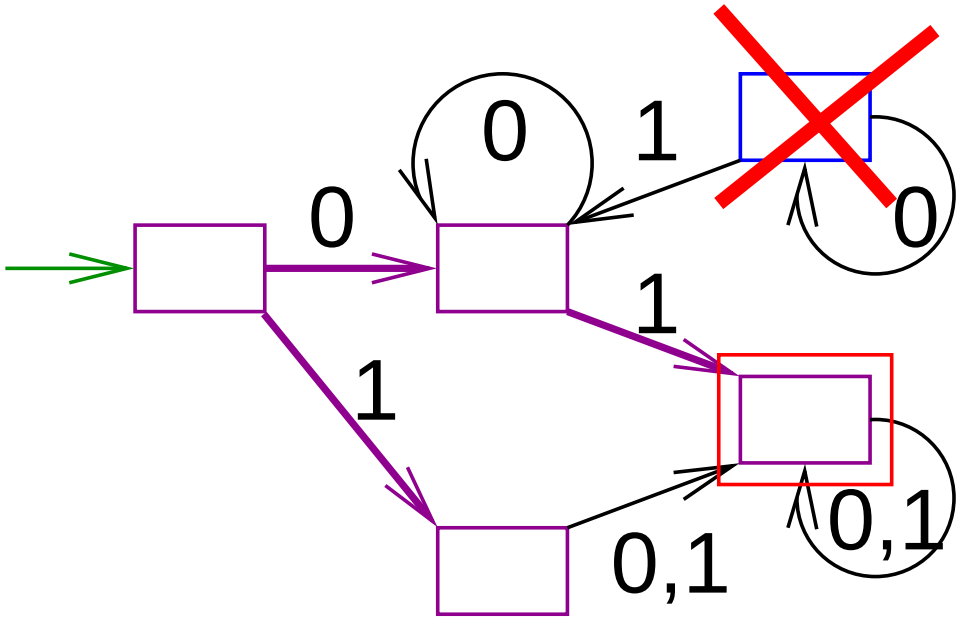
Remove the states, not reachable from s.

Algorithm: Depth-first search in Graph G_A for s.

Mark all reachable states.

Remove the non reachable states.

Preprocessing — Example



Kante im Tiefensuchbaum

erreichter Zustand

Equivalent states

Idea: consider DFA $M = (Q, \Sigma, \delta, s, F)$ (without non-reachable states)

M not state-minimal $\longrightarrow$

R_M refines R_L $\longrightarrow$

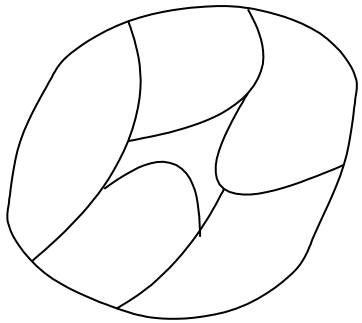
$\exists q \neq r \in Q$: Eq-class$(q) \dot{\cup}$ Eq-class$(r) \subseteq K$

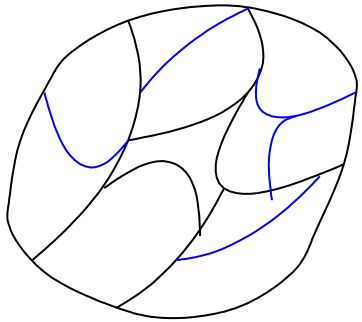
for an Eq-class K of R_L

such q, r are called equivalent, $q \equiv r$.

In particular:

$\forall w \in \Sigma^* : \hat{\delta}(q, w) \in F \Leftrightarrow \hat{\delta}(r, w) \in F$





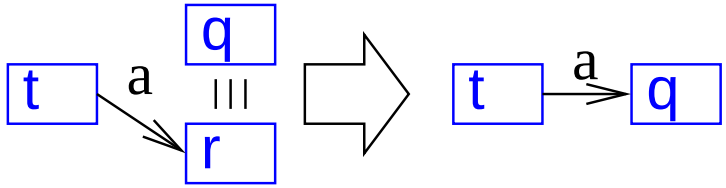
Remove equivalent states

Consider $q \neq r \in Q : q \equiv r$ with $r \neq s$

Remove r:

$M' := (Q \setminus \{r\}, \Sigma, \delta', s, F \setminus \{r\}))$ where

$$\delta'(t,a) := \begin{cases} q & \text{if } \delta(t,a) = r \\ \delta(t,a) & \text{otherwise} \end{cases}.$$



Lemma: $L(M') = L$

Proof: Exercise

State minimization

First step:

Function minDFA(M)

remove states not reachable from s

while $\exists q, r \in Q : q \equiv r \wedge q \neq r \wedge r \neq s$ **do**

remove r from M

return M

Problem: how to find the equivalent states?

$q \equiv r$ iff $\forall z \in \Sigma^* : \hat{\delta}(q,z) \in F \Leftrightarrow \hat{\delta}(r,z) \in F$

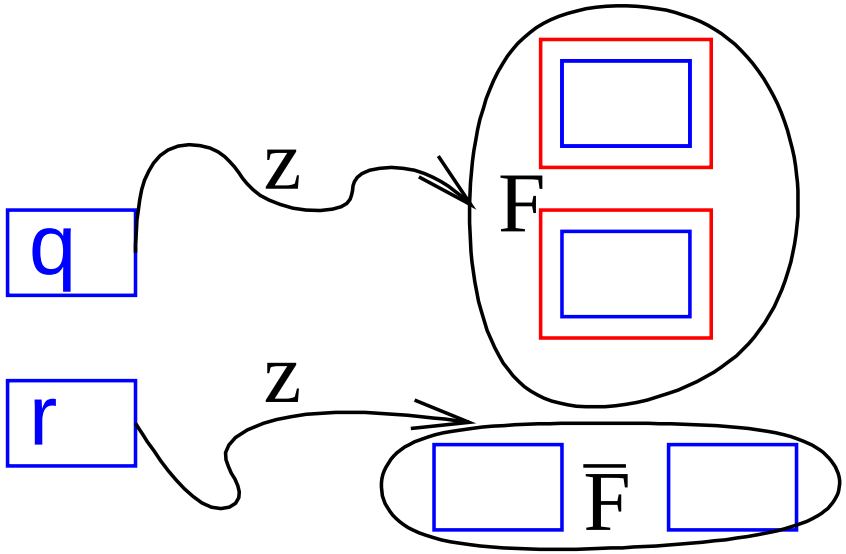
All quantifiers over not finite sets!

Nonequivalence for states

$q \equiv r$ iff $\forall z \in \Sigma^* : \hat{\delta}(q,z) \in F \Leftrightarrow \hat{\delta}(r,z) \in F$

$q \not\equiv r$ iff $\exists z \in \Sigma^* : \hat{\delta}(q,z) \in F \not\Leftrightarrow \hat{\delta}(r,z) \in F$

z testifies non-equivalence.



Problem: find the witnesses for non-equivalence

Shortest witnesses for non-equivalence

$\forall q \in F, r \notin F$: ε testifies $q \not\equiv r$.

Let $w = aw'$ be the shortest witness for $q \not\equiv r$.

Observation: w' is a witness for $q' := \delta(q,a) \not\equiv \delta(r,a) =: r'$

Lemma: w' is the shortest witness for $q' \not\equiv r'$

A proof by contradictions: Assume:

w'' is a shorter witness for $q' \not\equiv r'$

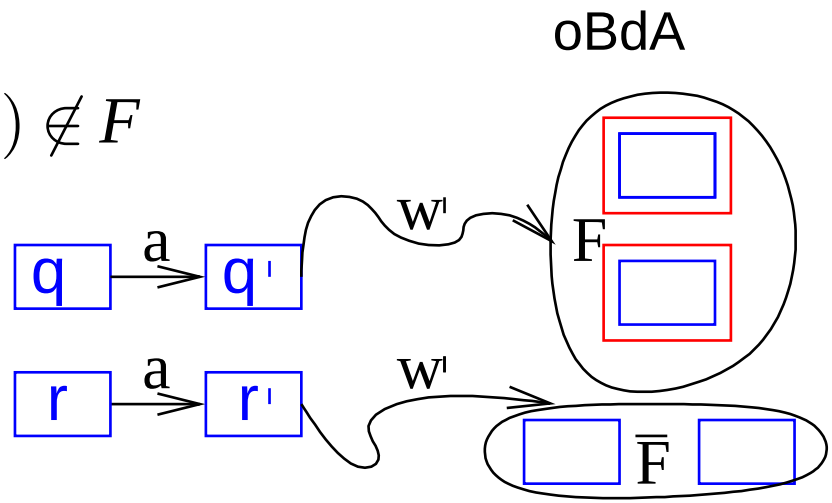
$\longrightarrow \hat{\delta}(q', w'') \in F \wedge \hat{\delta}(r', w'') \notin F$

$\longrightarrow \hat{\delta}(\delta(q,a), w'') \in F \wedge \hat{\delta}(\delta(r,a), w'') \notin F$

$\longrightarrow \hat{\delta}(q, aw'') \in F \wedge \hat{\delta}(r, aw'') \notin F$

$\longrightarrow aw''$ is a shorter witness for $q \not\equiv r$

A contradiction



Shortest witnesses for non-equivalence

ε testifies $q \not\equiv r$ if $q \in F, r \notin F$.

Let $w = aw'$ be the shortest witness for $q \not\equiv r$.

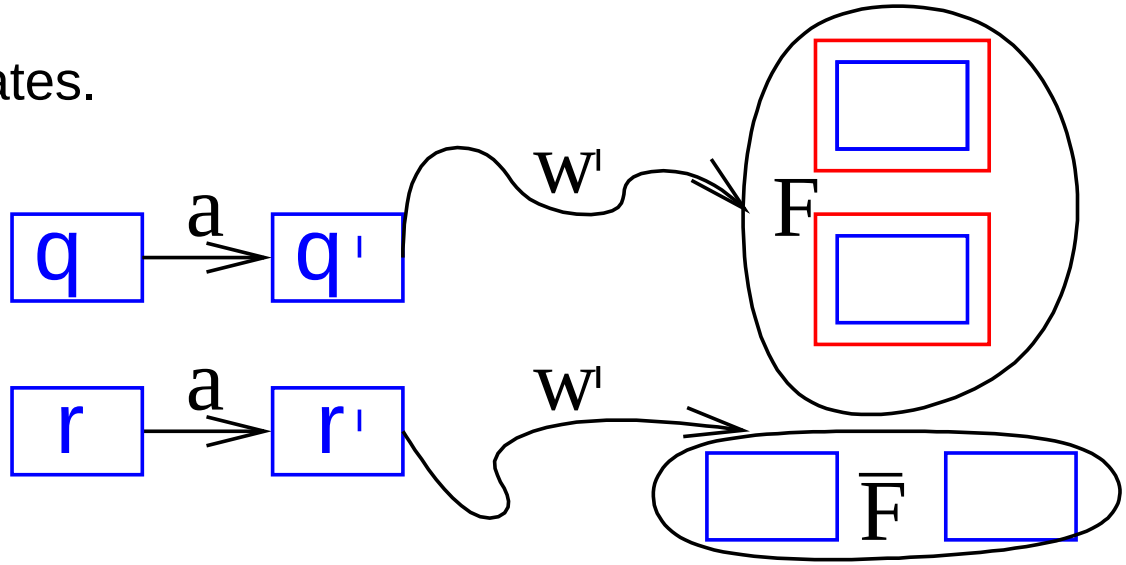
Lemma: w' is the shortest witness for $q' := \delta(q,a) \not\equiv \delta(r,a) =: r'$

Conclusion:

the shortest witnesses are found systematically and efficiently.

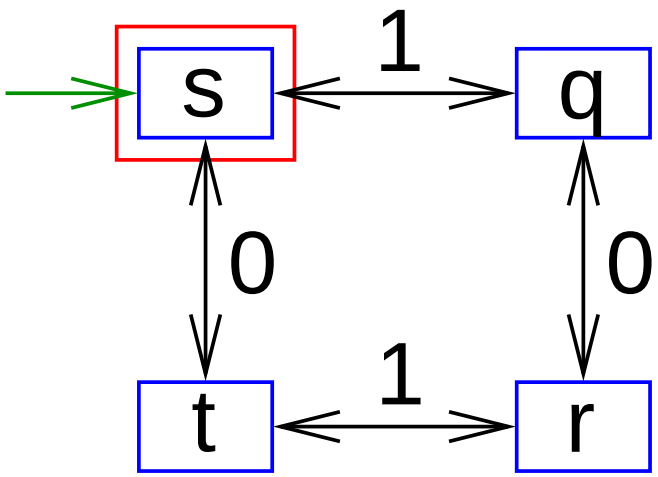
$\rightsquigarrow$ all non-equivalences

$\rightsquigarrow$ equivalence classes for states.



Example

$L \subseteq \{0,1\}^*$ language, all words with even number of one's and even number of zero's



$0 = 0\varepsilon$ is the shortest witness for $t \not\equiv r$.

$\rightsquigarrow \varepsilon$ is the shortest witness for $s = \delta(t,0) \not\equiv \delta(r,0) = q$.

Let $w = aw'$ be shortest witness for $q \not\equiv r$.

Lemma: w' is the shortest witness for $q' := \delta(q,a) \not\equiv \delta(r,a) =: r'$

Let N be the set all non-equivalent pairs of states for the witnesses with length $\leq k$.

For which pairs of states $\{q,r\}$ there are witnesses with length $k+1$?

Lemma: $\{\{q,r\} \subseteq Q : \exists a \in \Sigma : \{\delta(q,a), \delta(r,a)\} \in N\}$

An easier algorithm

$N := \emptyset$ // marked pairs

$N' := \{\{q,r\} \subseteq Q : q \in F \not\Leftrightarrow r \in F\}$ // pairs to be marked next

while $N' \neq \emptyset$ **do**

$N := N \cup N'$

$N' := \{\{q,r\} \subseteq Q : \exists a \in \Sigma : \{\delta(q,a), \delta(r,a)\} \in N\} \setminus N$

Total time: $\mathcal{O}\left(|\Sigma| \cdot |Q|^3\right)$

Initialization: $\mathcal{O}\left(|Q|^2\right)$

A loop time: $\mathcal{O}\left(|\Sigma| \cdot |Q|^2\right)$

How many loops? Surely $\leq |Q|^2$.

More exact observation: $\leq |Q|$ loops

State minimization in time $\mathcal{O}(|\Sigma| \cdot |Q| \log |Q|)$

[Hopcroft 1971]. Data structures.

Light simplification :

[Blum, Minimization of finite automata in $O(n \log n)$ time, Inf. Proc. Letters, 1996.]

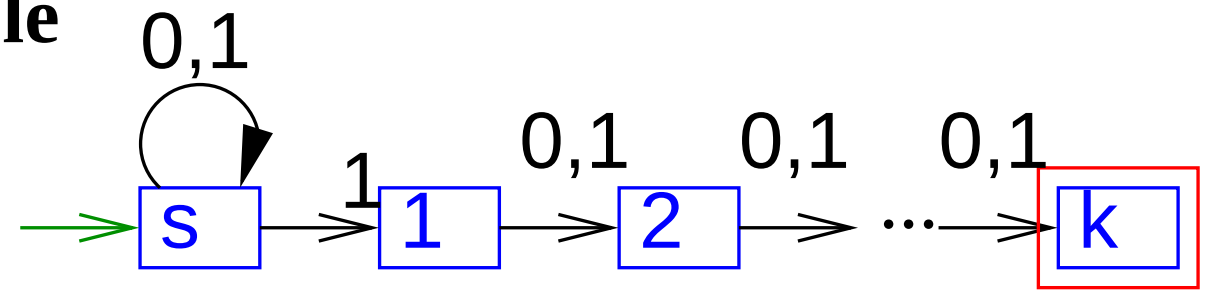
Why state minimization

- Minimal place for the the table of the transitions between states
- Minimal automaton obvious $\rightsquigarrow$ we learn something about the accepted language.

But, when δ is represented as a circuit or a program, we want to optimize its length and running time.

In general $\rightsquigarrow$ active research area.

Example

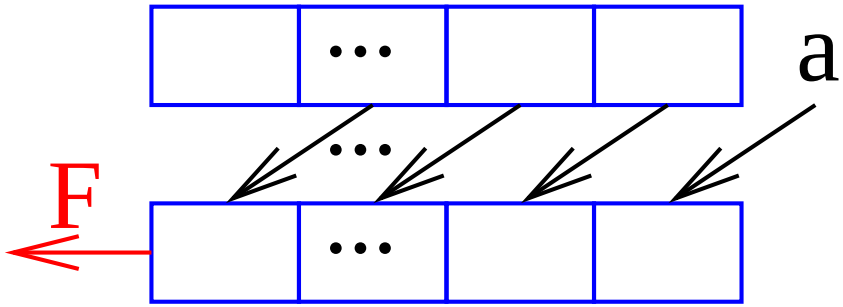


$$L = \{0,1\}^* 1 \{0,1\}^{k-1}$$

The minimal automaton has 2^k states.

$$(\{0,\ldots,2^k-1\}, \{0,1\}, \delta, 0, F)$$

$$\delta(q,a) = 2q + a,\ q \in F \Leftrightarrow q[k-1] = 1$$



Verification for non-regularity

- Pumping Lemma:
  - $+$: Easy manageable
  - $-$: Only necessarily condition
- Nerode relation
  - $+$: Necessary and sufficient condition $\text{index}(R_L) = \infty$
  - $-$: Somewhat unmanageable

Nerode relation

Example: $L = \{a^n b^n : n \geq 1\}$

Statement: $\forall k > 1, j \neq k > 1 : [a^k b] \neq [a^j b]$

$[a^k b] = \{a^k b, a^{k+1} bb, \ldots\} = \{a^{k+i} b^{i+1}\}$

so always $k-1$ more a's than b's.

Thus $[a^k b]$ and $[a^j b]$ are disjoint. □

Nerode relation

Example: $L = \{c^m a^\ell b^\ell : m, \ell \geq 0\} \cup \{a, b\}^*$

Statement: $\forall k > 1, j \neq k > 1 : [ca^k b] \neq [ca^j b]$

$[ca^k b] = \{c^m a^{k+i} b^{1+i} : m \geq 0, i \geq 1\}$

so always $k - 1$ more a's than b's.

Thus $[ca^k b]$ and $[ca^j b]$ are disjoint. □

1.2.6 Closure properties

Let L, L' be regular languages.

Then the following languages are regular:

$L \cup L', L^*, L \cdot L'$: by definition of the reg. expression.

$\overline{L} := \Sigma^* \setminus L$: Consider DFA $A = (Q, \Sigma, \delta, s, F)$ with $L(A) = L$.
Let $\overline{A} := (Q, \Sigma, \delta, s, Q \setminus F)$. Then $L(\overline{A}) = \overline{L}$.

$L \cap L' = \overline{\overline{L} \cup \overline{L'}}$ (De Morgan)

$L \setminus L' = L \cap \overline{L'}$

L^R: Exercise. Hint: Induction on the regular expressions.

Product automaton

Construction of a DFA for the set operations

L and L' are regular languages defined by DFAs

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

$A' = (Q', \Sigma, \delta', s', F')$.

Idea: An automaton $A_\times$ emulates the behavior of A and A'.

Product automaton: $A_\times := (Q \times Q', \Sigma, \delta_\times, (s, s'), F_\times)$ with

$\delta_\times((q, q'), a) = (\delta(q, a), \delta'(q', a))$

Define F in according to the set operation:

$L \cup L'$: $F_\times := Q \times F' \cup F \times Q'$

$L \cap L'$ $F_\times := F \times F'$

...

1.2.7 Decidability of some simple properties of a finite automaton

Word problem

$w \in L$?

Run in the DFA A.

Simulate A by the input w.

Is any final state reachable?

Linear time if DFA is known!

Emptiness problem

$L = \emptyset$?

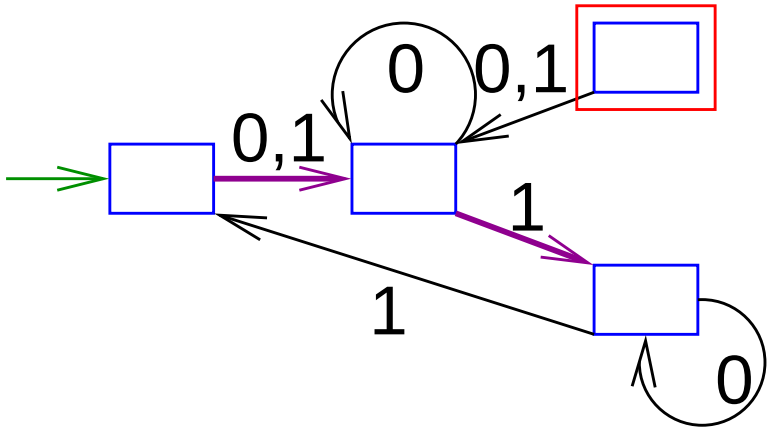
Representation of DFA or NFA A:

$L = \emptyset \Leftrightarrow \neg\exists f \in F : f$ is from s reachable

$\rightsquigarrow$ Depth-first search, linear time, as well as for NFA.

Example

$L(A) = \emptyset$



Kante im Tiefensuchbaum

Finiteness problem I — by Pumping Lemma

Let n be the number from Pumping-Lemma for L (type-3)

Statement: $|L(G)| = \infty \Leftrightarrow \exists z \in L(G) : n \leq |z| < 2n$

Proof:

$z \in L(G), n \leq |z| < 2n \longrightarrow$ Pumping lemma assures $|L| = \infty$.

If $|L(G)| = \infty$ consider $z \in L(G)$ with minimal $|z| \geq n$.

Assume $|z| \geq 2n$.

$\overset{\text{Pumping lemma}}{\longrightarrow} z = uvw, 1 \leq |v| \leq |uv| \leq n, uw \in L(G) \longrightarrow |uw| \geq n.$

A contradiction with the minimality of $|z|$.

Finiteness problem II — Cycle detection

$|L(A)| = \infty$? $\Leftrightarrow$ $\exists$ accepting path consisting a cycle.

Let NFA with $F = \{f\}$. Let $G_A = (Q, E)$,

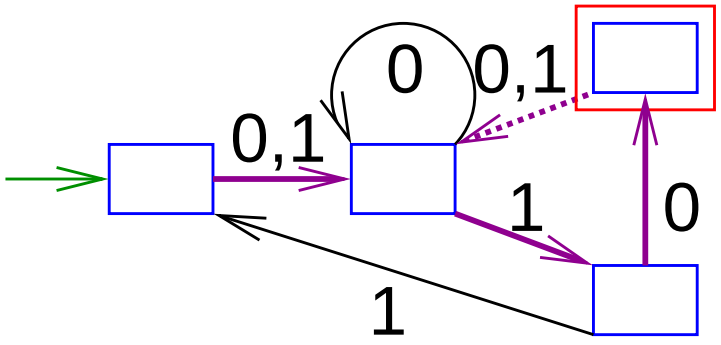
$E = \{(q,r) : \exists a \in \Sigma \cup \{\varepsilon\} : r \in \delta(q,a)\}$

1. Remove the states, from which f is not reachable.

   Depth-first search in $\overline{G_A} = (Q, \{(q,r) : (r,q) \in E\})$ for f.

2. Could we reach from s a cycle? $\Leftrightarrow$

   Does the depth-first search from s in G_A meet a backwards node?



Kante im Tiefensuchbaum

Rückwärtskante im Tiefensuchbaum

Completeness problem

$L(A) = \Sigma^*$?

$\Leftrightarrow \neg\exists q \in Q \setminus F$: q is from s reachable?

$\rightsquigarrow$ depth-first search, linear time, only for DFA!

(Equivalent: Emptiness of $\overline{L}$)

Completeness of NFA:

Transformation in DFA. No more obvious is known.

Equivalence problem

L and L' are regular languages defined by DFAs A, A'.

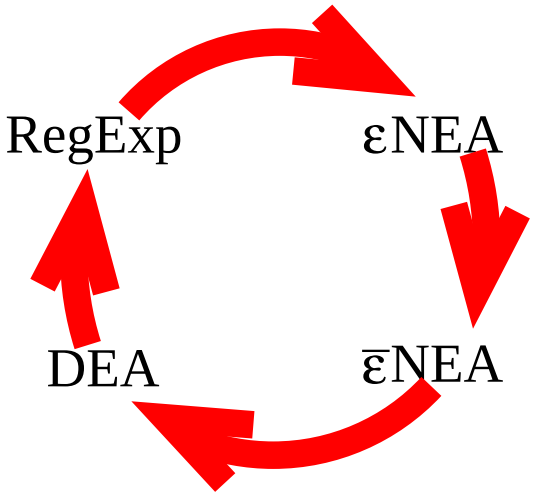
Question $L = L'$?

$\Leftrightarrow \neg\exists w : (w \in L \wedge w \notin L') \vee (w \notin L \wedge w \in L')$

$\Leftrightarrow \neg\exists w : (w \in L \wedge w \in \overline{L'}) \vee (w \in \overline{L} \wedge w \in L')$

$\Leftrightarrow (L \cap \overline{L'}) \cup (\overline{L} \cap L') = \emptyset$

for example via a product automaton



Problem: slowly

Equivalence for DFAs

L and L' are regular languages defined by DFAs $A = (Q, \Sigma, \delta, s, F)$, $A' = (Q', \Sigma, \delta', s', F')$.

Idea: the Minimal automaton is „unambiguous“.

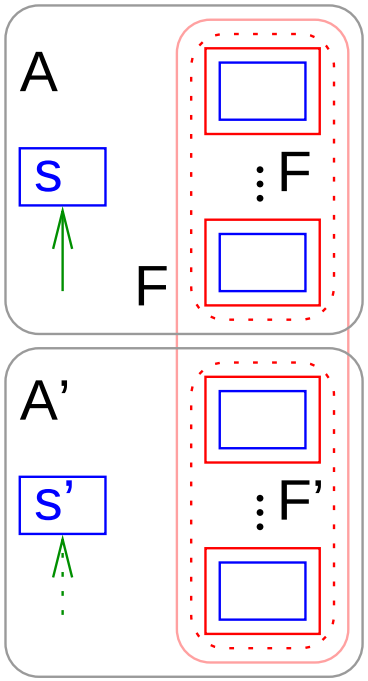
$\rightsquigarrow$ minimize both automata and prove the „equality“.

Problem: A renaming of the states is allowed. The complexity of isomorphism on more general graphs is an open problem.

Equivalence for DFAs

L and L' are regular languages defined by DFAs $A = (Q, \Sigma, \delta, s, F)$, $A' = (Q', \Sigma, \delta', s', F')$. Let $Q \cap Q' = \emptyset$.

Question: $L = L'$?



Consider $A_\cup := (Q \cup Q', \Sigma, \delta_\cup, s, F \cup F')$, where

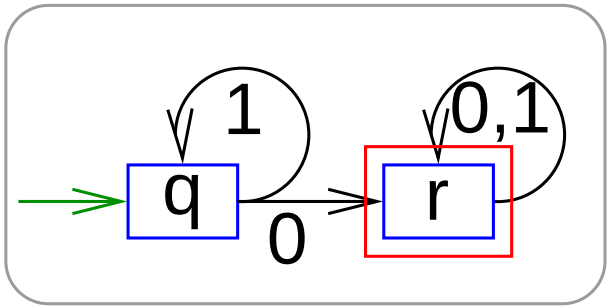
$$\delta_\cup(q,a) = \begin{cases} \delta(q,a) & \text{if } q \in Q \\ \delta'(q,a) & \text{if } q \in Q' \end{cases}$$

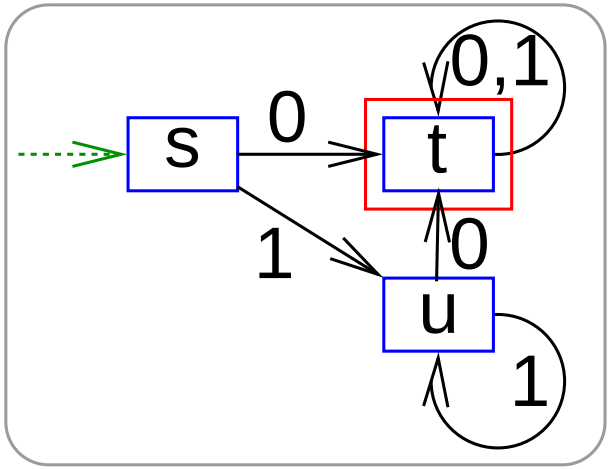
Find the equivalence classes of the states for $A_\cup$.

$L = L' \Leftrightarrow s \equiv s'$.

Example

$L \subseteq \{0,1\}^*$ language, all words with at least one zero





Algorithm for marking all

inequivalent pairs of states supplies:

$\{q,r\}, \{q,t\}, \{s,r\}, \{s,t\}, \{u,r\}, \{u,t\}$

$\rightsquigarrow q \equiv s$

$\rightsquigarrow$ Both automata are equivalent.

Summary of finite automata and regular languages

- The most simple machine model
- Involving algorithmic rules (State minimization, converting with regular expressions,. . .)
- Accepted languages completely understandable
- Useful applications: text manipulation, compilers, hardware,. . .
- Concept of a nondeterminism