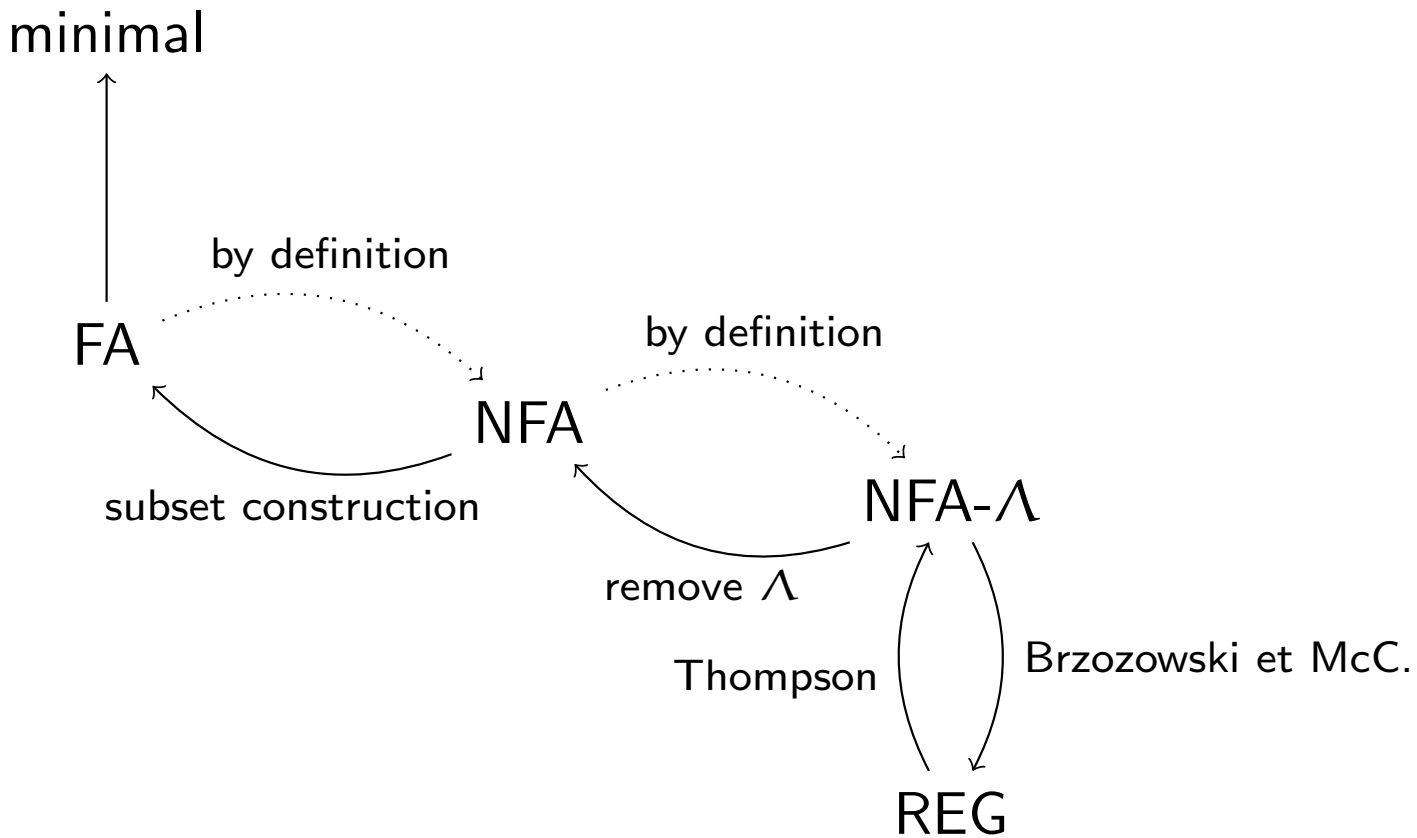

minimal
by definition
FA
by definition
NFA
subset construction
NFA-Λ
remove Λ
Thompson
Brzozowski et McC.
REG

Definition (REG)

– $\varnothing$ is in REG.

– $\{a\}$ in REG, for every $a \in \Sigma$

– if L_1 and L_2 in REG,
then so are $L_1 \cup L_2$, $L_1 \cdot L_2$, and L_1^*.

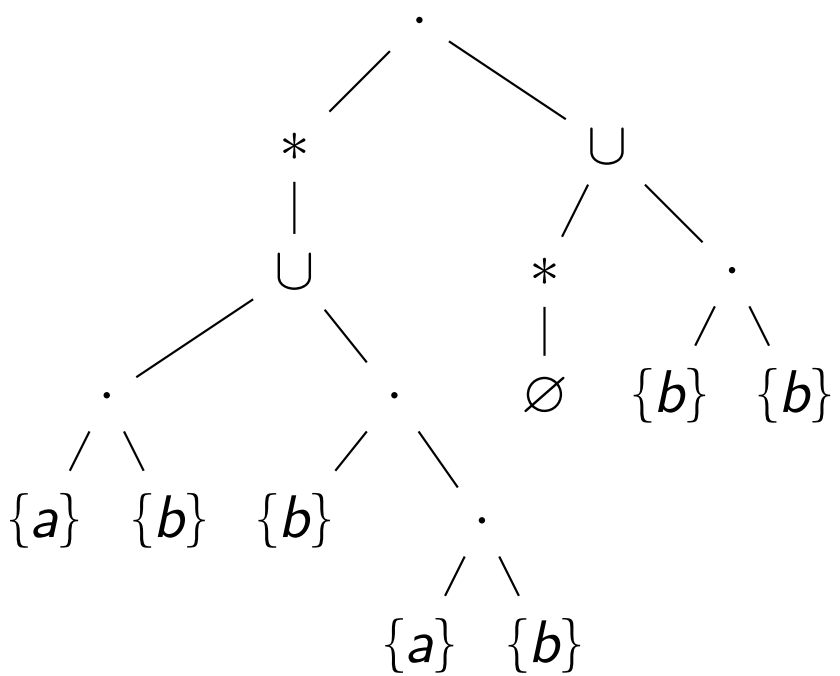
[M] D. 3.1 $\mathcal{R}$

Smallest set[family] of languages that
– contains $\varnothing$ and $\{a\}$ for $a \in \Sigma$, and — basis
– is *closed under* union, concatenation and star. — induction

[M] cf. E 1.20

$\{ab, bab\}^*\{\Lambda, bb\}$

$(\ (\{a\} \cdot \{b\}) \cup (\{b\} \cdot \{a\} \cdot \{b\}) \ )^* \cdot (\ \varnothing^* \cup (\{b\} \cdot \{b\}) \ )$



– $\varnothing$, Λ, and a are RegEx (for all $a \in \Sigma$)
– if E_1 and E_2 are RegEx, then so are E_1^*, $(E_1 + E_2)$, and $(E_1 E_2)$

expression [syntax] vs its language [semantics]

E string	$L(E)$ language
$\varnothing$	$\varnothing$
Λ	$\{\Lambda\}$
a	$\{a\}$
$(E_1 + E_2)$	$L(E_1) \cup L(E_2)$
$(E_1 E_2)$	$L(E_1) \cdot L(E_2)$
E_1^*	$L(E_1)^*$

we say

$E_1 = E_2$ iff $L(E_1) = L(E_2)$

– Odd number of a

$bba_0\,ba_1\,bbba_2\,bba_1\,a_2\,bb$

[M] E 3.2

– Odd number of a

$bba_0ba_1bbba_2bba_1a_2bb$

$b^*ab^*(ab^*a)^*b^*$	not correct
$b^*ab^*(ab^*ab^*)^*$	$b^*ab^*(ab^*ab^*)(ab^*ab^*)$
$b^*a(b^*ab^*ab^*)^*$	not correct
$b^*a(b^*ab^*a)^*b^*$	$b^*a(b^*ab^*a)(b^*ab^*a)b^*$
$b^*a(b+ab^*a)^*$	$b^*ab^*(ab^*a)b^*(ab^*a)b^*$

[M] E 3.2

– Ending with b, no aa

$bb(ab)bbb(ab)(ab)b$

$(b + ab)^*(b + ab)$ at least once

[M] cf. E 3.3, see ↪E. 2.3

– No aa may also end in a

$(b + ab)^*(\Lambda + a)$

– Even number of a and even number of b

two letters together

aa and bb keep both numbers even [odd]

ab and ba switch between even and odd, for both numbers

– Even number of a and even number of b

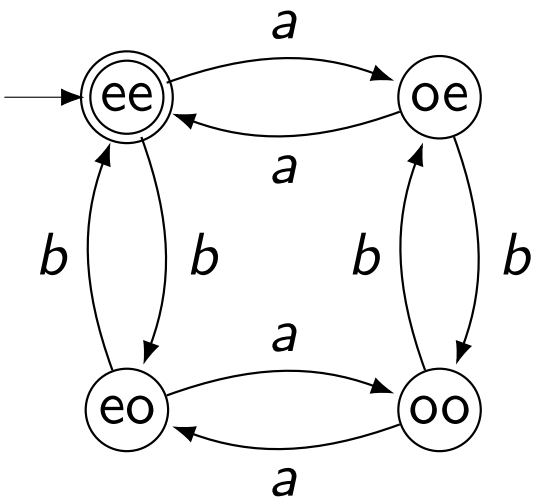
two letters together

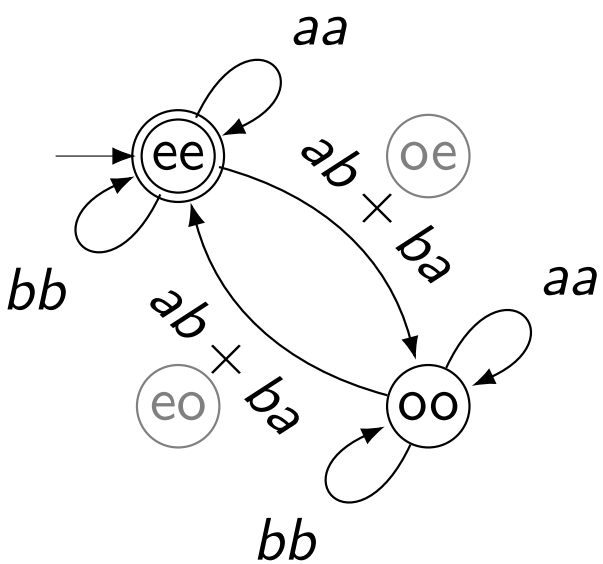
aa and bb keep both numbers even [odd]

ab and ba switch between even and odd, for both numbers

$$(aa + bb + (ab + ba)(aa + bb)^*(ab + ba))^*$$

[M] E 3.4, see ↪Brzozowski *et* McCluskey





– Numeric constants in programming language

14, +1, -12, 14.3, -.99, 16., 3E14, -1.00E2, 4.1E-1, .3E+2

[M] E 3.5

– Numeric constants in programming language

14, +1, -12, 14.3, -.99, 16., 3E14, -1.00E2, 4.1E-1, .3E+2

Use d for $(0+1+2+3+4+5+6+7+8+9)$

Use s for $(\Lambda + \text{'+'} + \text{'-'})$

Use p for '.'

$$s(dd^*(\Lambda + pd^*) + pdd^*)(\Lambda + Esdd^*)$$

[M] E 3.5

Exercise

$\Sigma = \{\ell, ., @\}$, where ℓ represents a letter

In pairs, come up with a regular expression for email addresses, e.g., *automata@liacs.leidenuniv.nl* or *firstname.lastname@example.nl*

$\Sigma = \{\ell, ., @\}$, where ℓ represents a letter

In pairs, come up with a regular expression for email addresses, e.g., *automata@liacs.leidenuniv.nl* or *firstname.lastname@example.nl*

$$\ell\ell^*(.\ell\ell^*)^*@\ell\ell^*(.\ell\ell^*)(.\ell\ell^*)^*$$

$\approx$[Stanford CS103] L15

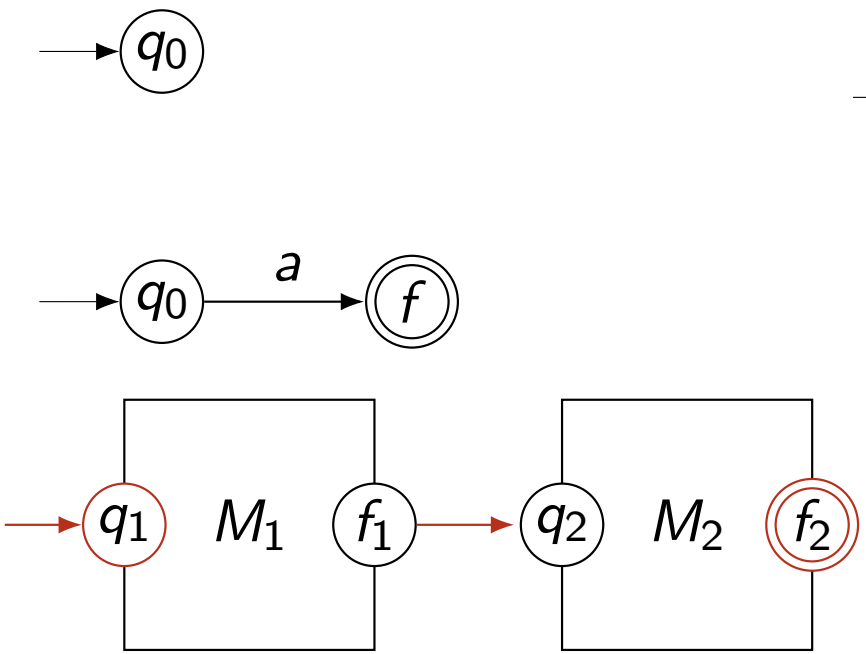
Theorem (Kleene)

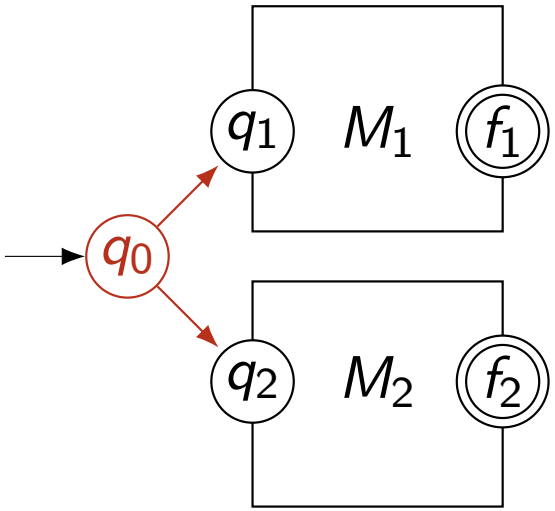
Finite automata and regular expressions specify the same family of languages.

- from RegEx to FA
  $\hookrightarrow$Thompson's construction
- from FA to RegEx
  $\hookrightarrow$McNaughton and Yamada indicated by $\star$, discussed in class only if we have time, otherwise please go through if you are interested
  State elimination $\hookrightarrow$Brzozowski et McCluskey

Theorem

If L is a regular language, then there exists an NFA that accepts L.



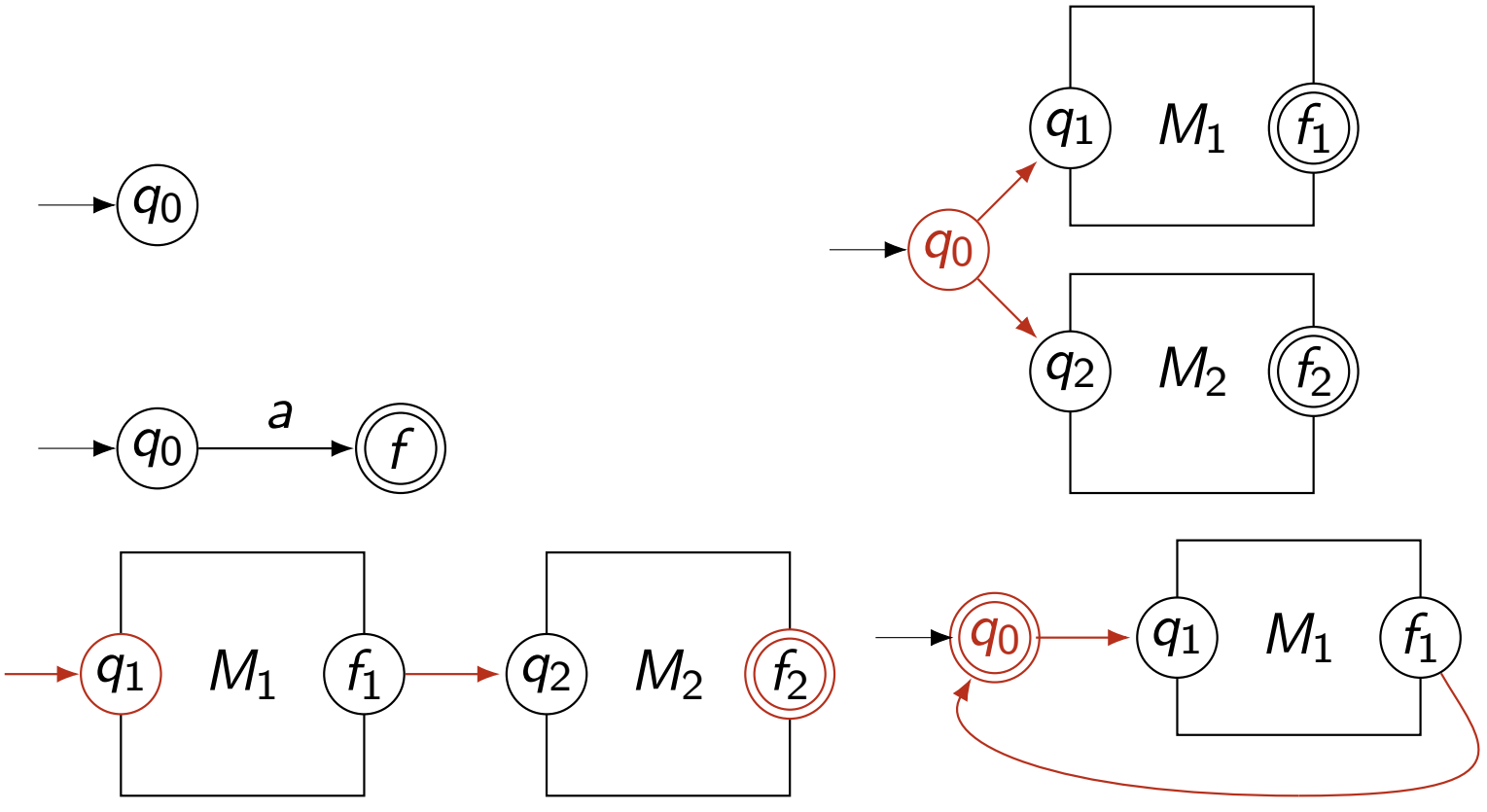


NFA for L^*...

[M] Th 3.25 [L] Th 3.1

Theorem

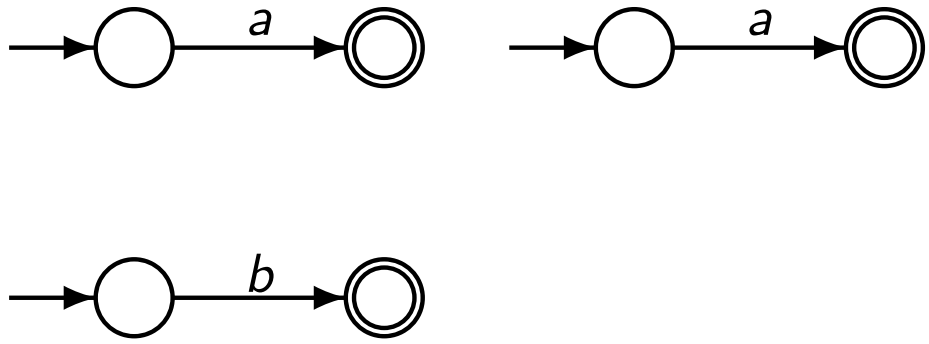
If L is a regular language, then there exists an NFA that accepts L.

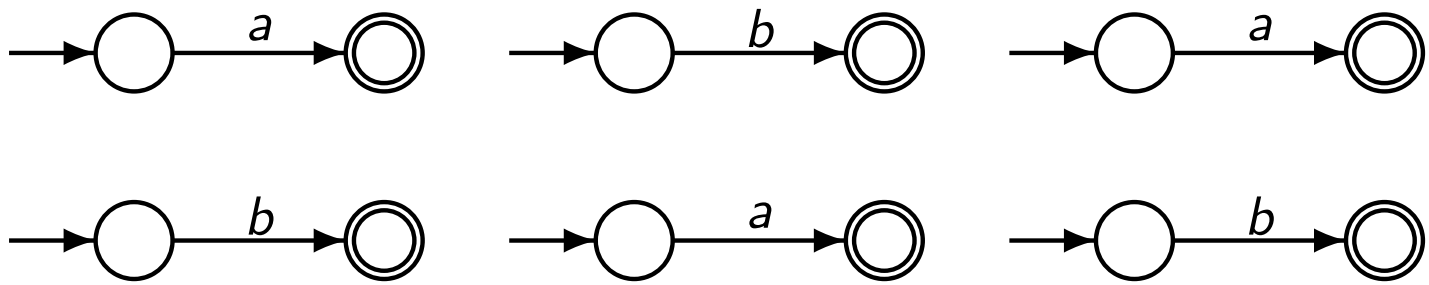


[M] Th 3.25 [L] Th 3.1

Example 3.28. An NFA Corresponding to $((aa + b)^*(aba)^*bab)^*$

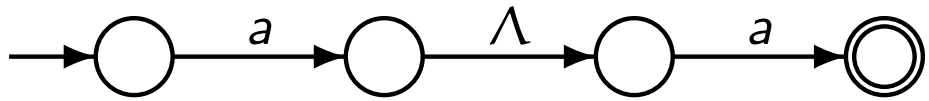
Step 1

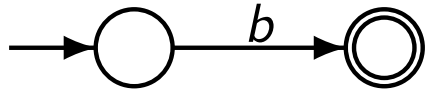


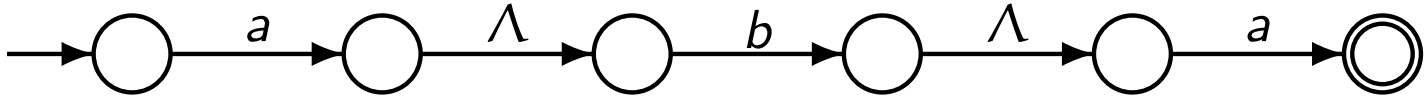


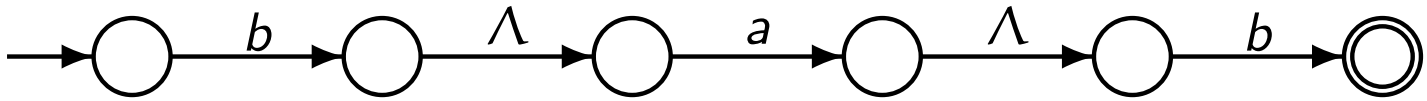
Example 3.28. An NFA Corresponding to $((aa + b)^*(aba)^*bab)^*$

Step 2



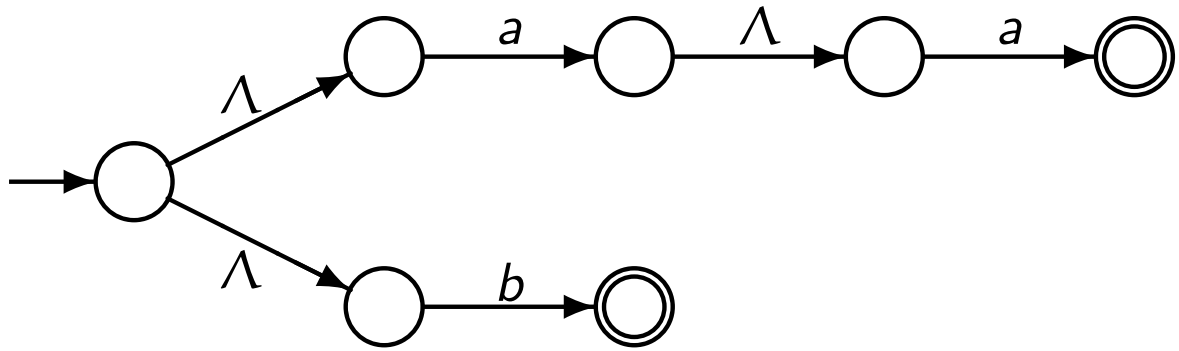


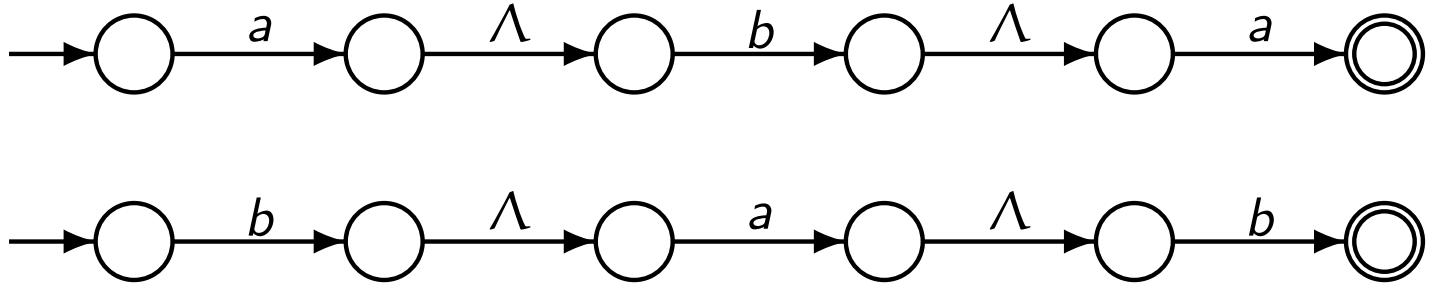




Example 3.28. An NFA Corresponding to $((aa + b)^*(aba)^*bab)^*$

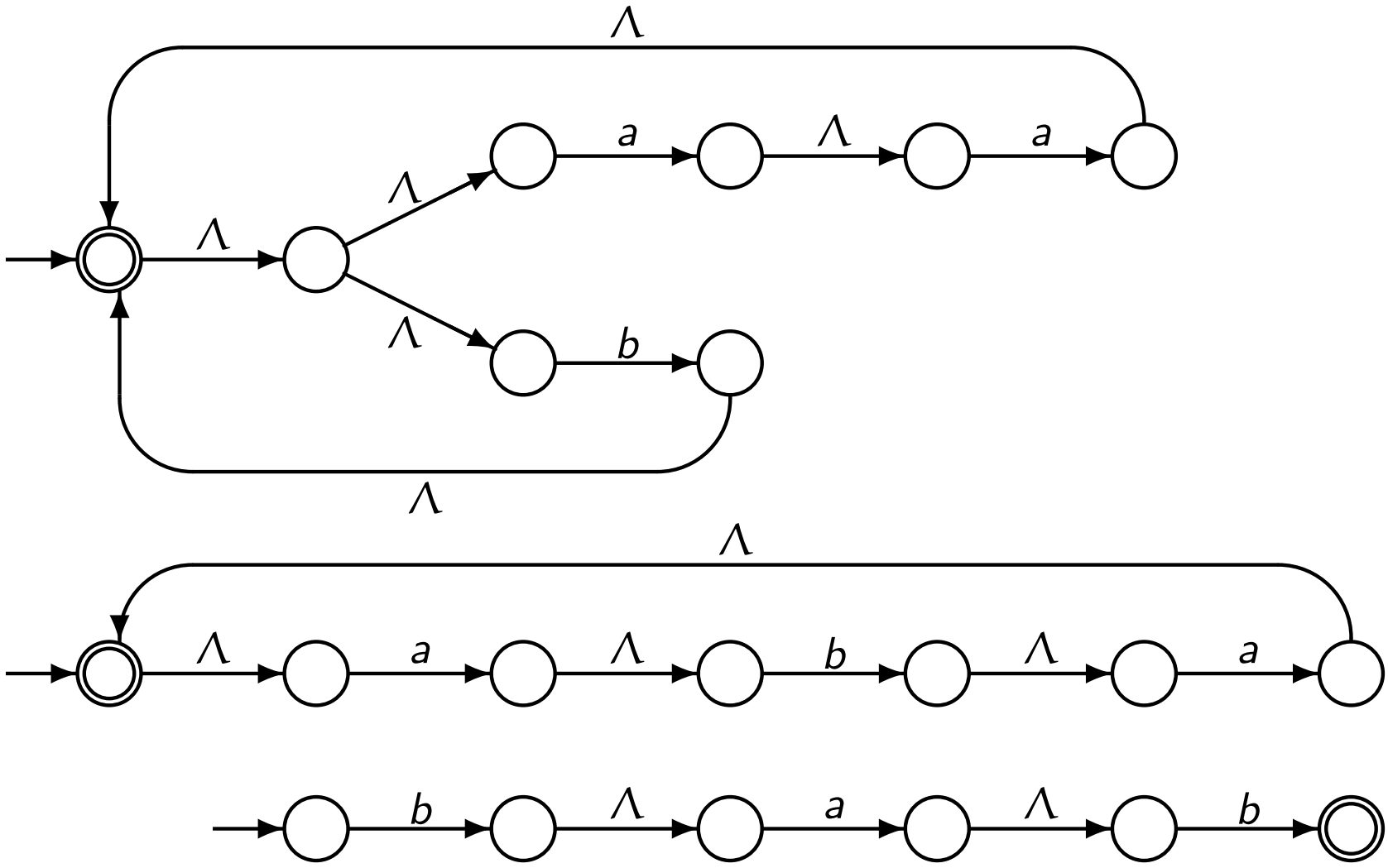
Step 3





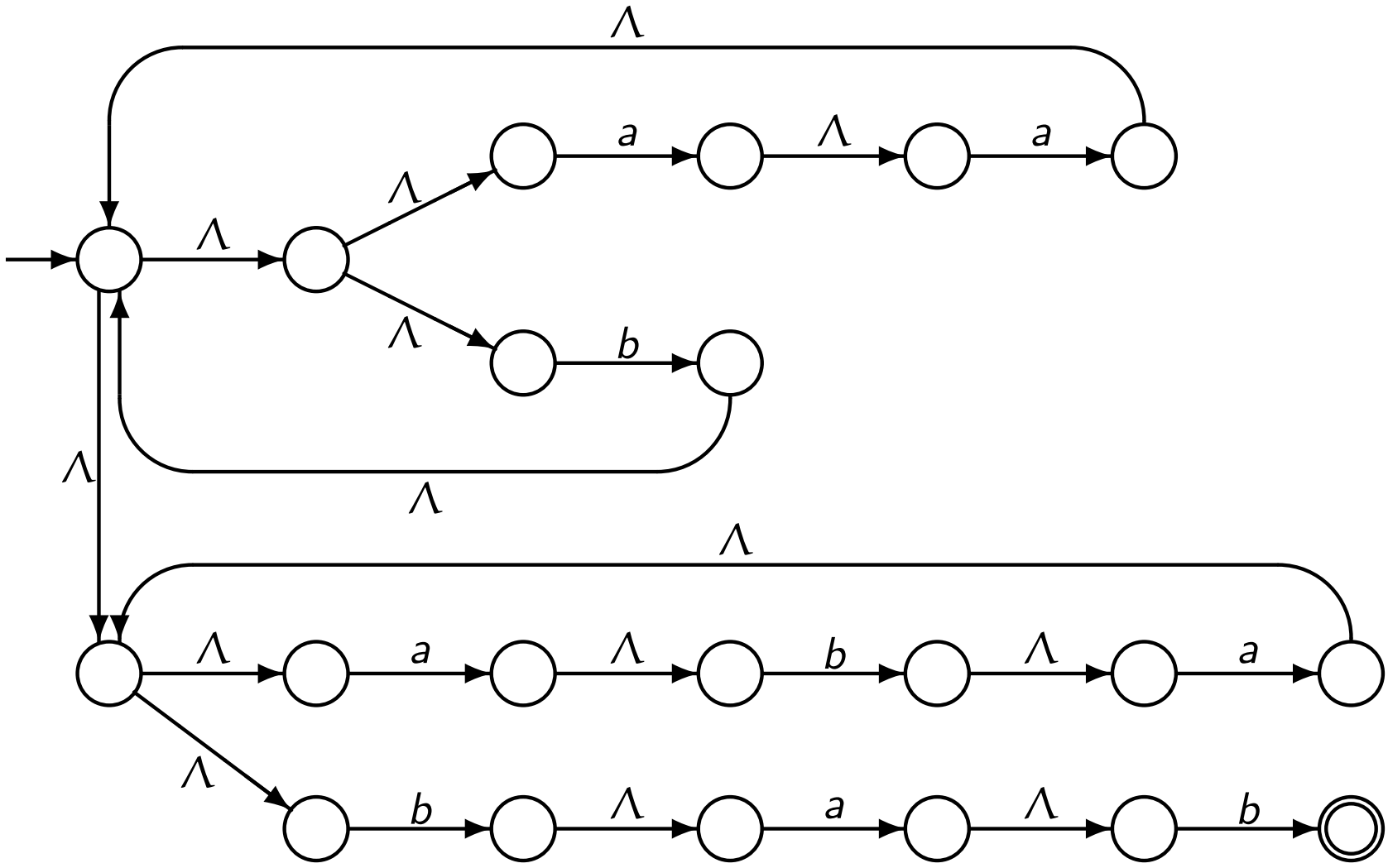
Example 3.28. An NFA Corresponding to $((aa + b)^*(aba)^*bab)^*$

Step 4



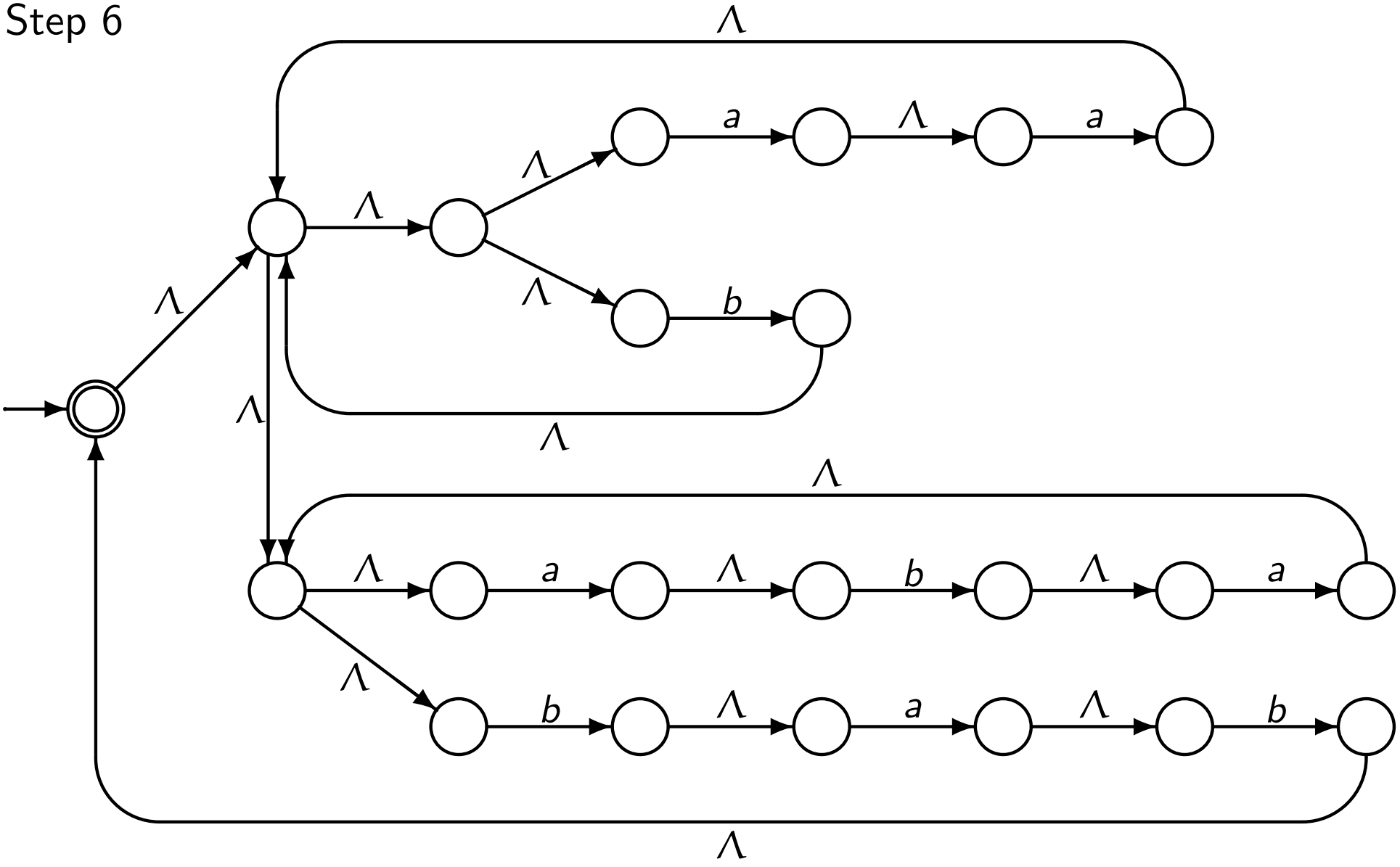
Example 3.28. An NFA Corresponding to $((aa + b)^*(aba)^*bab)^*$

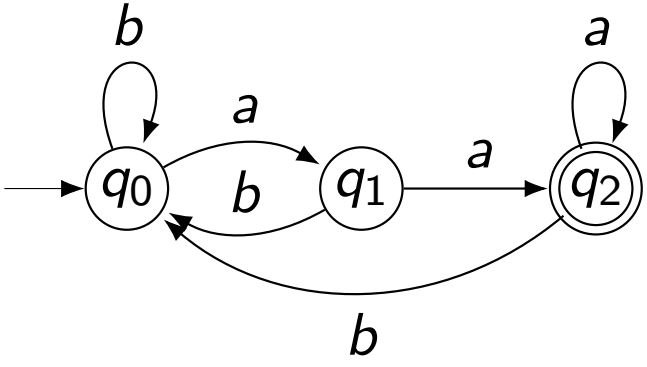
Step 5



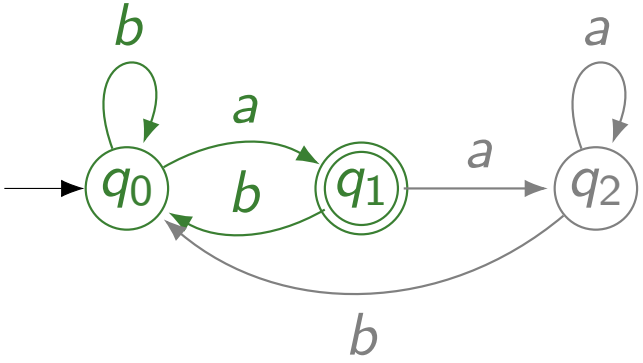
Example 3.28. An NFA Corresponding to $((aa + b)^*(aba)^*bab)^*$

Step 6

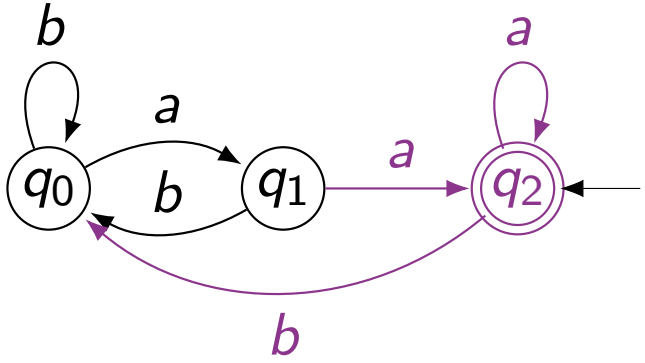



b
a
a
a
b
q0
q1
q2
b

Intro: finding a regular expression



$\underbrace{(b + ab)^* a}_{\text{loop on } q_0}$



$\underbrace{b\,[(b + ab)^* a]\,a + a}_{\text{single loop on } q_2}$

$$\underbrace{[(b + ab)^* aa]}_{\text{from } q_0 \text{ to } q_2}\ \underbrace{[b(b + ab)^* aa + a]^*}_{\text{loop on } q_2}$$

short answer $(a + b)^* aa$ see $\hookrightarrow$FA example

ABOVE

It is possible to construct an expression for a small automaton "by hand" by starting with a restricted version of the automaton, and slowly adding nodes and edges.

BELOW

Next a formal proof how this can be done generally, referred to as the McNaughton–Yamada algorithm.

The expression is built iteratively. First we consider only paths in the automaton that can not pass any node: we only consider single edges. Then we add the nodes one by one. Regular expression $r^k(i,j)$ includes all strings from paths from i to j that only pass by nodes from 1 to k. (We always may exit or enter any other node, but only as first or last node of the path.)

LATER

The method of Brzozowski and McCluskey below "implements" this proof, using a generalized automaton. It features graphs with edges that carry regular expressions.

Theorem

If M is an FA, then $L(M)$ is regular.

PROOF

$M = (Q, \Sigma, q_0, A, \delta)$ assume $Q = \{1, 2, \ldots, n\}$ $q_0 = 1$

$L^k(i, j)$ only paths $i, p_1, \ldots, p_\ell, j$ with $1 \leqslant p_\ell \leqslant k$

[M] Th 3.30

cf. Floyd's algorithm for all-pairs shortest path problem

Theorem

If M is an FA, then $L(M)$ is regular.

PROOF

$M = (Q, \Sigma, q_0, A, \delta)$ assume $Q = \{1, 2, \ldots, n\}$ $q_0 = 1$

$L^k(i, j)$ only paths $i, p_1, \ldots, p_\ell, j$ with $1 \leqslant p_\ell \leqslant k$

$L^0(i, j) = \{a \mid \delta(i, a) = j\}$ $i \neq j$ basis

$L^0(i, j) = \{a \mid \delta(i, a) = j\} \cup \{\Lambda\}$ $i = j$

[M] Th 3.30

Theorem

If M is an FA, then $L(M)$ is regular.

PROOF

$M = (Q, \Sigma, q_0, A, \delta)$ assume $Q = \{1, 2, \ldots, n\}$ $q_0 = 1$

$L^k(i, j)$ only paths $i, p_1, \ldots, p_\ell, j$ with $1 \leqslant p_\ell \leqslant k$

$L^0(i, j) = \{a \mid \delta(i, a) = j\}$ $i \neq j$ basis

$L^0(i, j) = \{a \mid \delta(i, a) = j\} \cup \{\Lambda\}$ $i = j$

one by one add nodes, k from 1 to n:

$L^k(i, j) = \ldots$

[M] Th 3.30

Theorem

If M is an FA, then $L(M)$ is regular.

PROOF

$M = (Q, \Sigma, q_0, A, \delta)$ assume $Q = \{1, 2, \ldots, n\}$ $q_0 = 1$

$L^k(i, j)$ only paths $i, p_1, \ldots, p_\ell, j$ with $1 \leqslant p_\ell \leqslant k$

$L^0(i, j) = \{a \mid \delta(i, a) = j\}$ $i \neq j$ basis

$L^0(i, j) = \{a \mid \delta(i, a) = j\} \cup \{\Lambda\}$ $i = j$

one by one add nodes, k from 1 to n:

$$L^k(i, j) = L^{k-1}(i, j) \cup \underbrace{L^{k-1}(i, k)}_{\text{from } i \text{ to } k} \cdot \underbrace{\left(L^{k-1}(k, k) \right)^*}_{\text{loop from } k \text{ to } k} \cdot \underbrace{L^{k-1}(k, j)}_{\text{from } k \text{ to } j}$$

[M] Th 3.30

Theorem

If M is an FA, then $L(M)$ is regular.

PROOF

$M = (Q, \Sigma, q_0, A, \delta)$ assume $Q = \{1, 2, \ldots, n\}$ $q_0 = 1$

$L^k(i, j)$ only paths $i, p_1, \ldots, p_\ell, j$ with $1 \leqslant p_\ell \leqslant k$

$L^0(i, j) = \{a \mid \delta(i, a) = j\}$ $i \neq j$ basis

$L^0(i, j) = \{a \mid \delta(i, a) = j\} \cup \{\Lambda\}$ $i = j$

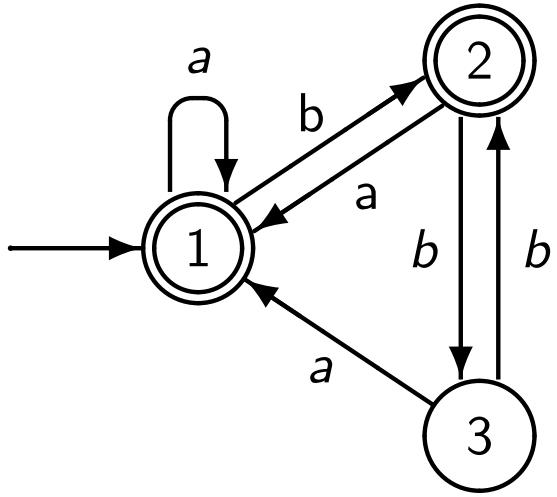
one by one add nodes, k from 1 to n:

$$L^k(i, j) = L^{k-1}(i, j) \cup \underbrace{L^{k-1}(i, k)}_{\text{from } i \text{ to } k} \cdot \underbrace{\left(L^{k-1}(k, k) \right)^*}_{\text{loop from } k \text{ to } k} \cdot \underbrace{L^{k-1}(k, j)}_{\text{from } k \text{ to } j}$$

$$L(M) = \bigcup_{j \in A} L^n(1, j)$$ full language, all nodes

[M] Th 3.30

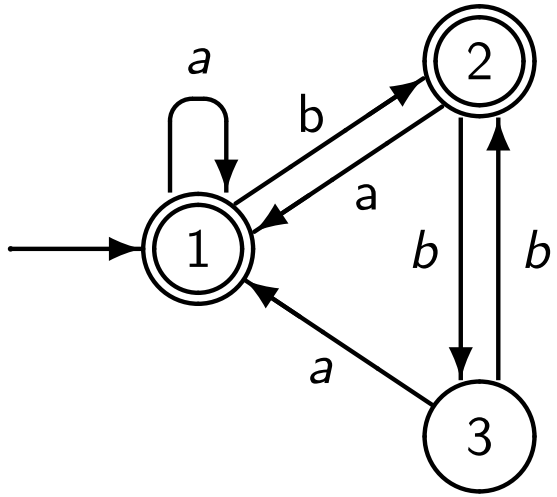
$r^k(i,j)$ expression for $L^k(i,j)$



$r^0(i,j)$	$j=1$	2	3
$i=1$	$a+\Lambda$	b	$\varnothing$
2	a	Λ	b
3	a	b	Λ

[M] E 3.32

$r^k(i,j)$ expression for $L^k(i,j)$



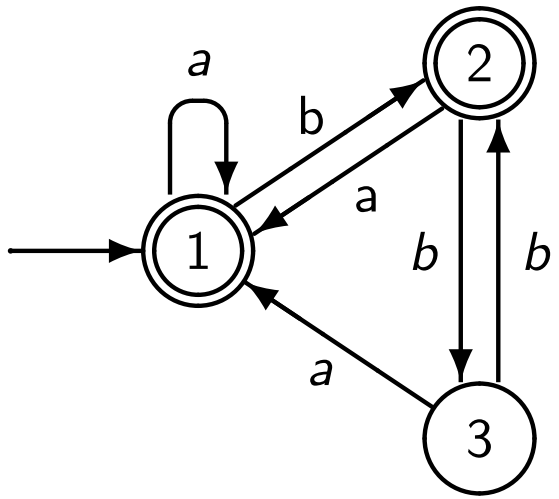
$r^0(i,j)$	$j=1$	2	3
$i=1$	$a+\Lambda$	b	$\varnothing$
2	a	Λ	b
3	a	b	Λ

$r^1(i,j)$	$j=1$	2	3
$i=1$	a^*	a^*b	$\varnothing$
2	aa^*	$\Lambda+aa^*b$	b
3	aa^*	a^*b	Λ

Simplified

[M] E 3.32

$r^k(i,j)$ expression for $L^k(i,j)$

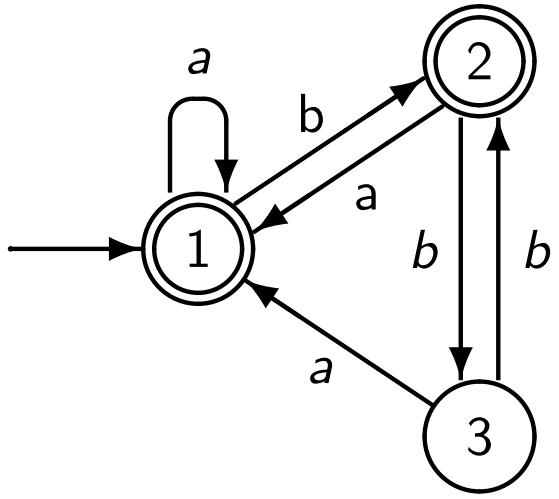


$r^1(i,j)$	$j=1$	2	3
$i=1$	a^*	a^*b	$\varnothing$
2	aa^*	$\Lambda + aa^*b$	b
3	aa^*	a^*b	Λ

$r^2(i,j)$	$j=1$	2	3
$i=1$	$a^*(baa^*)^*$	$a^*(baa^*)^*b$	$a^*(baa^*)^*bb$
2	$aa^*(baa^*)^*$	$(aa^*b)^*$	$(aa^*b)^*b$
3	$aa^* + a^*baa^*(baa^*)^*$	$a^*b(aa^*b)^*$	$\Lambda + a^*b(aa^*b)^*b$

Simplified / rewritten

$r^k(i,j)$ expression for $L^k(i,j)$



$r^2(i,j)$	$j=1$	2	3
$i=1$	$a^*(baa^*)^*$	$a^*(baa^*)^*b$	$a^*(baa^*)^*bb$
2	$aa^*(baa^*)^*$	$(aa^*b)^*$	$(aa^*b)^*b$
3	$aa^* + a^*baa^*(baa^*)^*$	$a^*b(aa^*b)^*$	$\Lambda + a^*b(aa^*b)^*b$

$$r^3(1,1) = r^2(1,1) + r^2(1,3)r^2(3,3)^*r^2(3,1)$$
$$r^3(1,2) = r^2(1,2) + r^2(1,3)r^2(3,3)^*r^2(3,2)$$

[M] E 3.32

BELOW The *state elimination method* by Brzozowski et McCluskey constructs a regular expression for a given automaton, by iteratively removing the states. The edges of the automaton do not just contain symbols (or Λ) but regular expressions themselves. Thus the graphs are a hybrid form of finite automata and regular expressions. It is rather clear however what they express.

Start by adding a new initial and accepting state; connect the initial state to the old initial state, and connect the old accepting states to the new accepting state, using as label the expression Λ (representing the empty word).

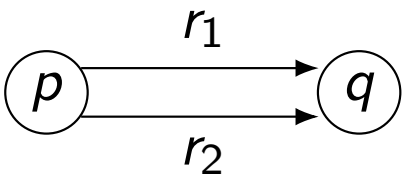
Whenever during this construction two parallel edges (p, r_1, q) and (p, r_2, q) appear, we replace them with a single edge $(p, r_1 + r_2, q)$

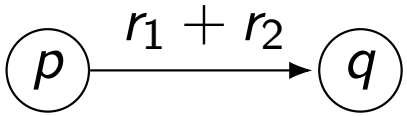
Choose any node q to be removed. Let r_2 be the expression on the loop for q. (If there is no loop we consider this expression to be $\varnothing$.)

For any incoming edge (p, r_1, q) and outgoing edge (q, r_3, s) we add the edge $(p, r_1 r_2^* r_3, s)$ which replace the path from p to s via q.

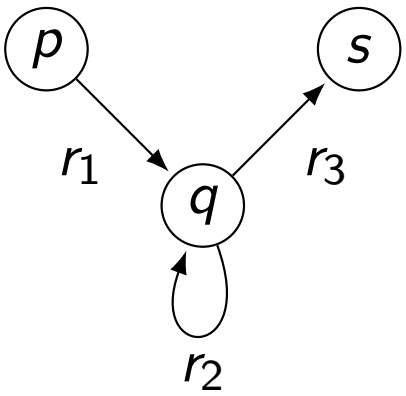
Remove q. Repeat.

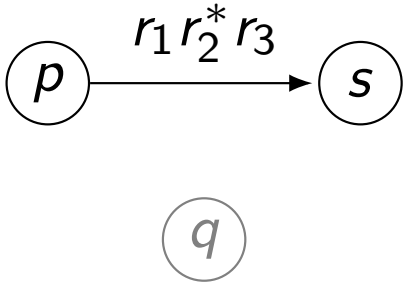
When all original nodes are removed, we obtain a graph with single edge; its label represents the language of the original automaton.



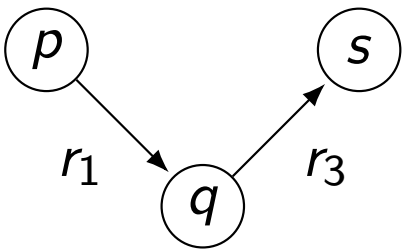


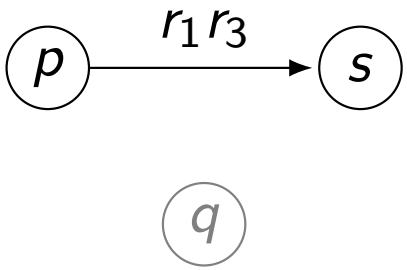
join parallel edges





reduce node q





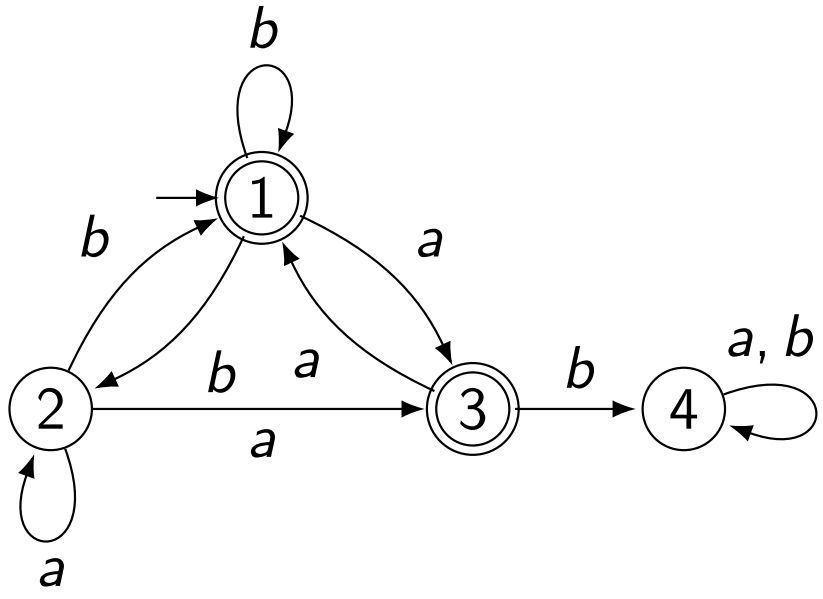
special case: $r_2 = \varnothing$

[M] Exercise 3.54

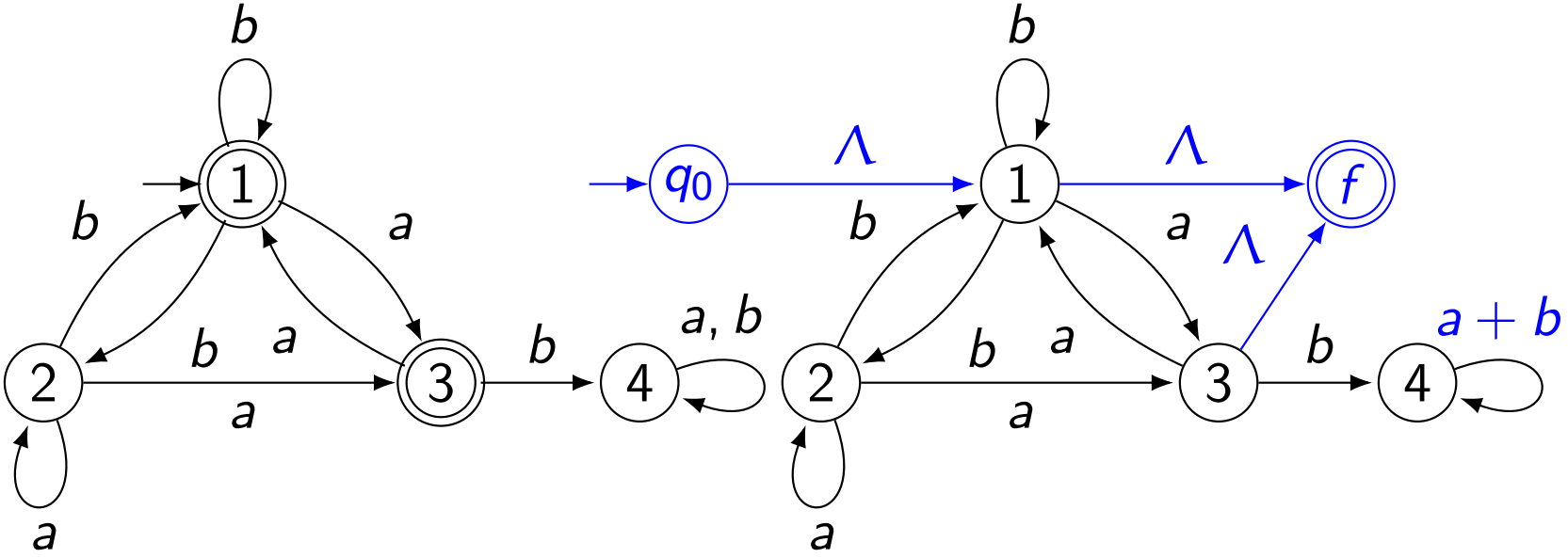
References

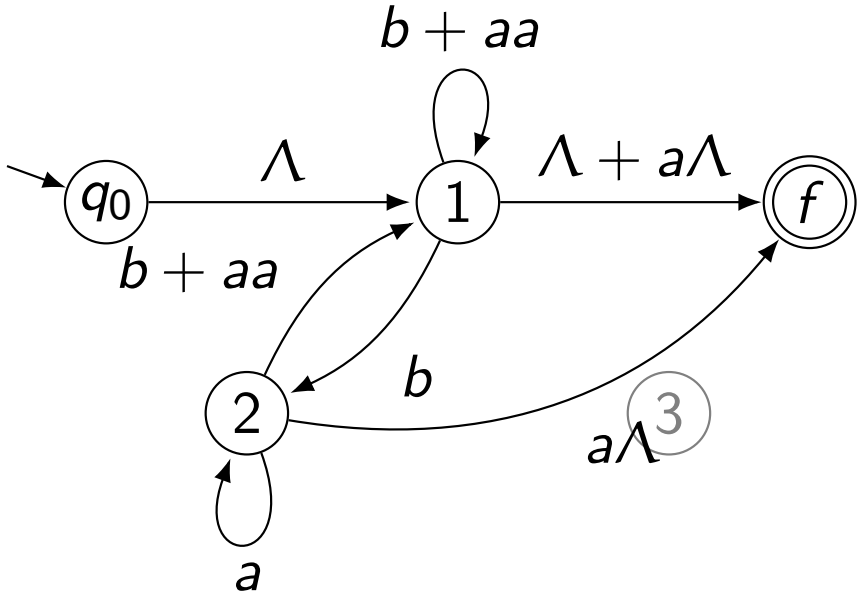
R. McNaughton and H. Yamada, Regular expressions and state graphs for automata, IRE Trans. Electronic Computers, vol. 9 (1960), 39–47.
S.C. Kleene. Representation of Events in Nerve Nets and Finite Automata. Automata Studies, Annals of Math. Studies. Princeton Univ. Press. 34 (1956)

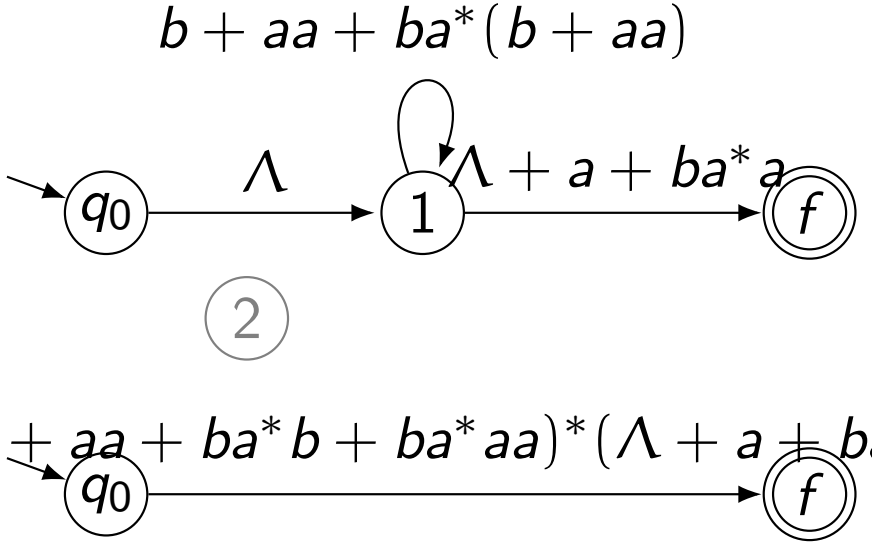
State elimination method:
J.A. Brzozowski et E.J. McCluskey, Signal Flow Graph Techniques for Sequential Circuit State Diagrams, IEEE Transactions on Electronic Computers, Institute of Electrical & Electronics Engineers (IEEE), vol. EC-12, no 2, avril 1963, p. 67–76. doi:10.1109/pgec.1963.263416



Eliminate 4,3,2,1


b
1
b
a
2
b
a
3
b
4
a, b
a
q0
Λ
1
Λ
f
b
a
Λ
2
b
a
3
b
4
a + b
a


b + aa
q0
Λ
1
Λ + aΛ
f
b + aa
2
b
aΛ
3
a


b + aa + ba*(b + aa)
q0
Λ
1
Λ + a + ba*a
f
2
(b + aa + ba*b + ba*aa)*(Λ + a + ba*a)
q0
f

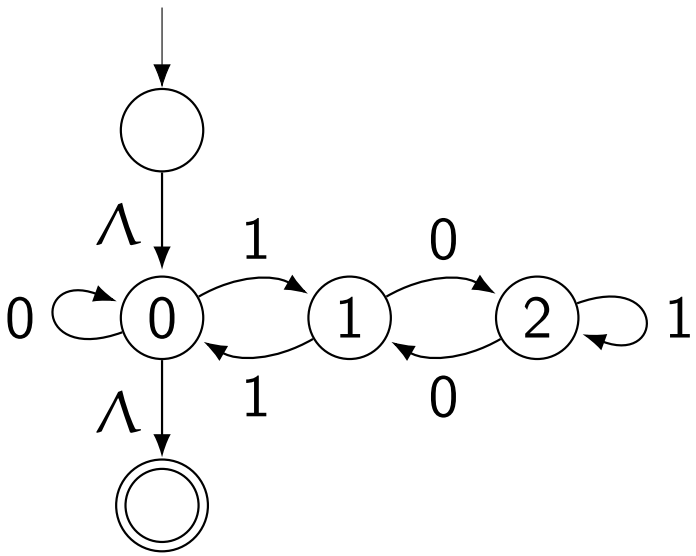
ABOVE

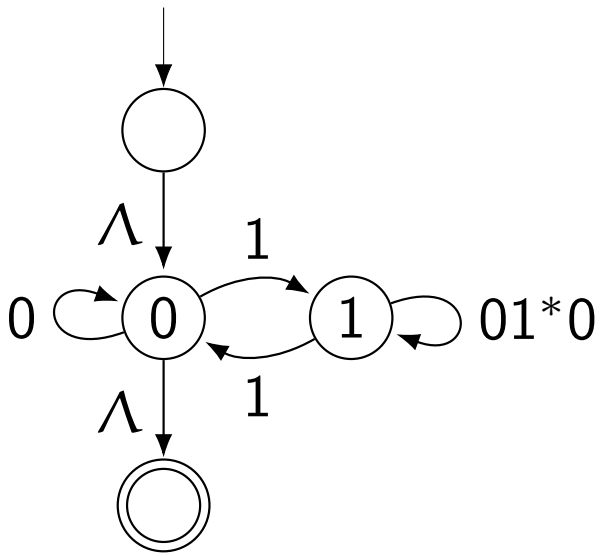
Start by adding new initial and accepting states i and f. Connect these to the original initial and accepting states by edges with the expression Λ.

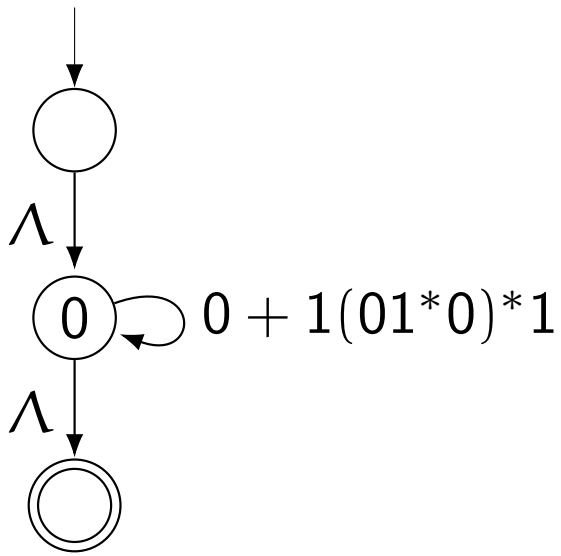
Note we also replaced the parallel edges a, b (loops on node 4) with the expression $a + b$.

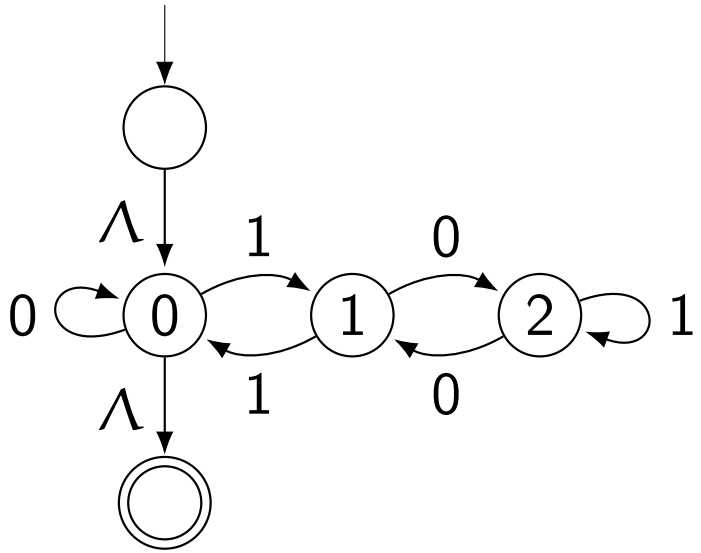
The first node that is eliminated is 4. The proces is not visible here, as there are no pairs (i, j) such that there are edges $(i, R_1, 4)$ and $(4, R_2, j)$, because there are no outgoing edges from 4. Thus no edges are constructed.

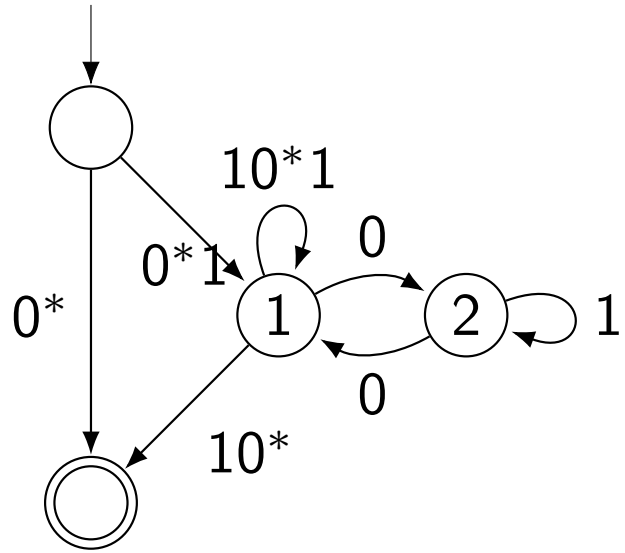
The second node eliminated is 3, as shown.

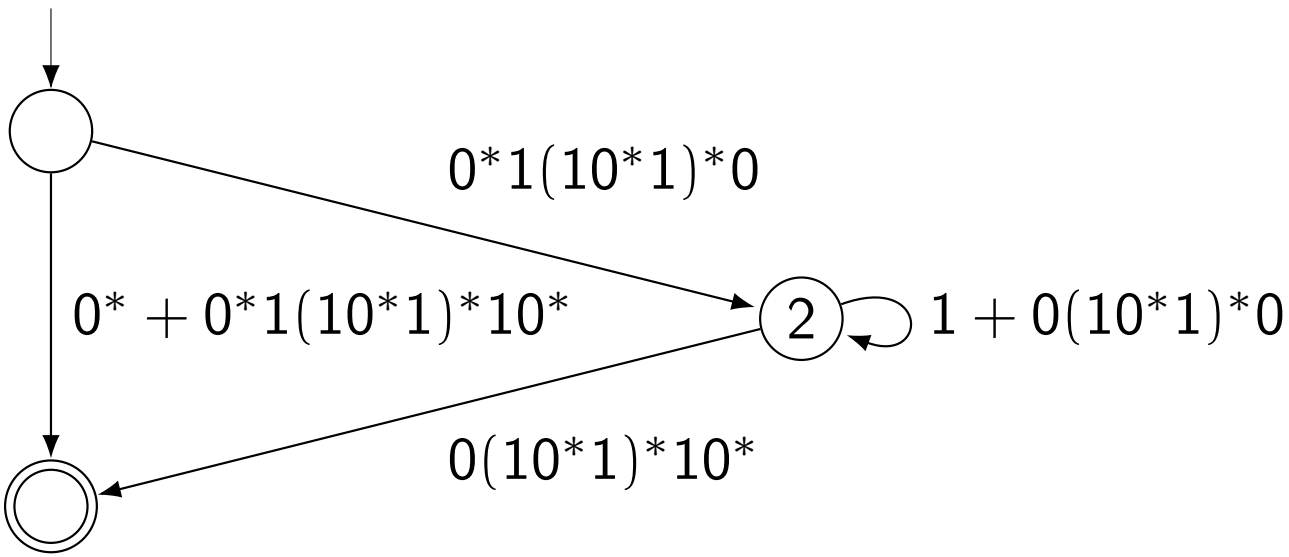

Λ
1
0
0
0
1
2
1
1
0
Λ

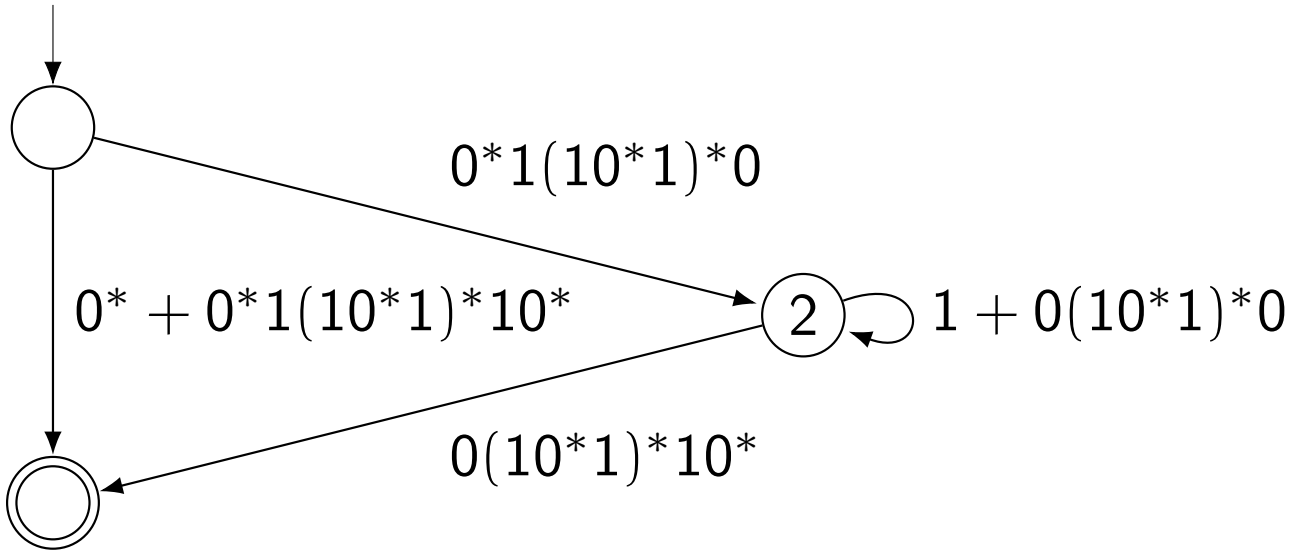

Λ
1
0
0
1
01*0
1
Λ

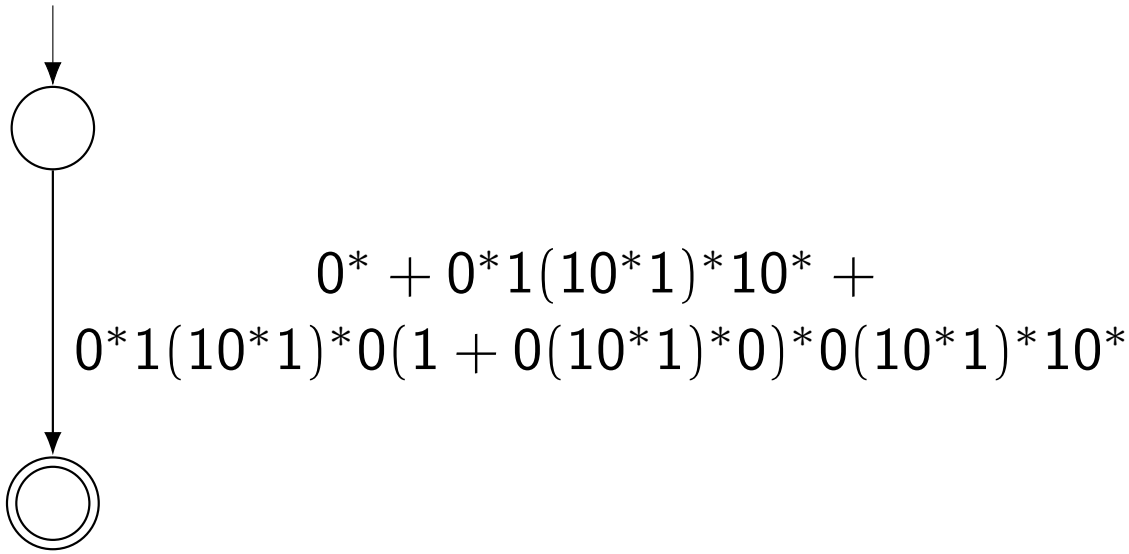

Λ
0
0 + 1(01*0)*1
Λ


Λ
1
0
0
0
1
2
1
Λ
1
0


10*1
0*1
0
0*
1
2
1
0
10*


0*1(10*1)*0
0* + 0*1(10*1)*10*
2
1 + 0(10*1)*0
0(10*1)*10*


$0^*1(10^*1)^*0$
$0^* + 0^*1(10^*1)^*10^*$
2
$1 + 0(10^*1)^*0$
$0(10^*1)^*10^*$


$0^* + 0^*1(10^*1)^*10^* +$
$0^*1(10^*1)^*0(1 + 0(10^*1)^*0)^*0(10^*1)^*10^*$

ABOVE

We compute a regular expression from the given automaton in two different reduction orders.

The first example reduces nodes in the order $2, 1, 0$. The result is $(0 + 1(01^*0)^*1)^*$
(The removal of the last loop is not depicted.)

The second example in the order $0, 1, 2$. The result is $0^* + 0^*1(10^*1)^*10^* + 0^*1(10^*1)^*0(1 + 0(10^*1)^*0)^*0(10^*1)^*10^*$
The result differs in structure and size.

$h : \Sigma_1 \to \Sigma_2^*$ letter-to-string map

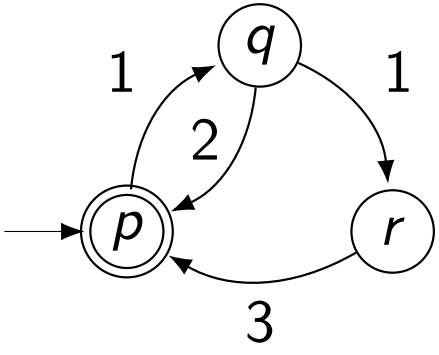
$$h : \begin{array}{ccc} 1 & \mapsto & aa \\ 2 & \mapsto & \Lambda \\ 3 & \mapsto & abb \end{array}$$

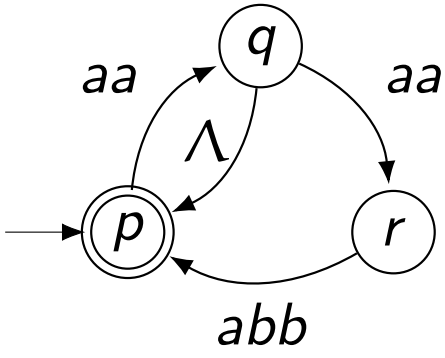
$h : \Sigma_1^* \to \Sigma_2^*$ string-to-string map

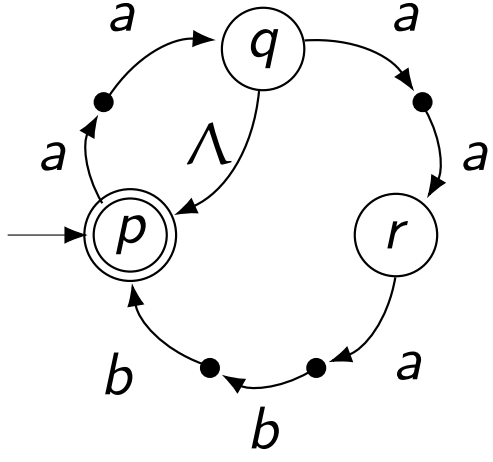
$h(\sigma_1\sigma_2 \ldots \sigma_k) = h(\sigma_1)h(\sigma_2) \ldots h(\sigma_k)$ $\quad h(12113) = aa \cdot \Lambda \cdot aa \cdot aa \cdot abb$

$K \subseteq \Sigma_1^*$ language-to-language map

$h(K) = \{\, h(x) \mid x \in K \,\}$



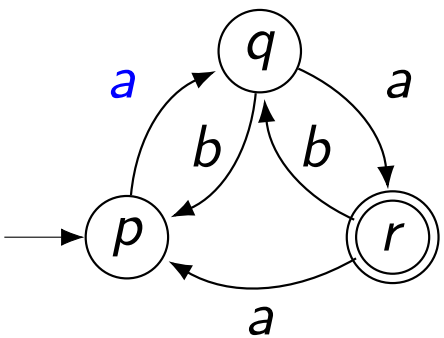


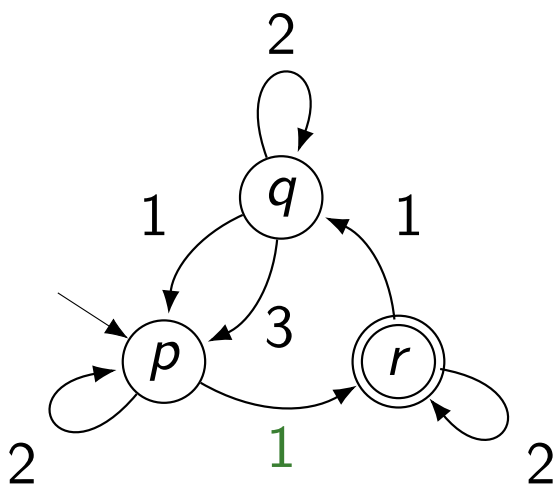


$h : \Sigma_1 \to \Sigma_2^*,\ L \subseteq \Sigma_2^* \qquad h^{-1}(L) = \{\, x \in \Sigma_1^* \mid h(x) \in L \,\}$

$$h: \begin{array}{ccc} 1 & \mapsto & aa \\ 2 & \mapsto & \Lambda \\ 3 & \mapsto & abb \end{array}$$

$$\begin{array}{cccccccccc} \Sigma_1^* & h^{-1}(L) & \ni & 1 & 2 & 1 & 3 & 1 \\ \downarrow h & & & & & & & \\ \Sigma_2^* & L & \ni & aa & \Lambda & aa & abb & aa \end{array}$$





Regular languages are closed under

– Boolean operations (complement, union, intersection, minus)
– Regular operations (union, concatenation, star)
– Reverse (mirror)
– [inverse] Homomorphism

In the wild

- software engineering, e.g., automata-based modeling language
- modelling of hardware circuits, a book on this
- lexical analysis (compiling high-level language program), e.g., lex
- networks protocols, e.g., TCP/IP, or in packet filtering BGP
- efficient string matching algorithm, e.g., Thompson's algorithm is used in `grep` in Unix
- buttons?
- other thoughts
- ...

Homework 2 is available!