# Automata Theory

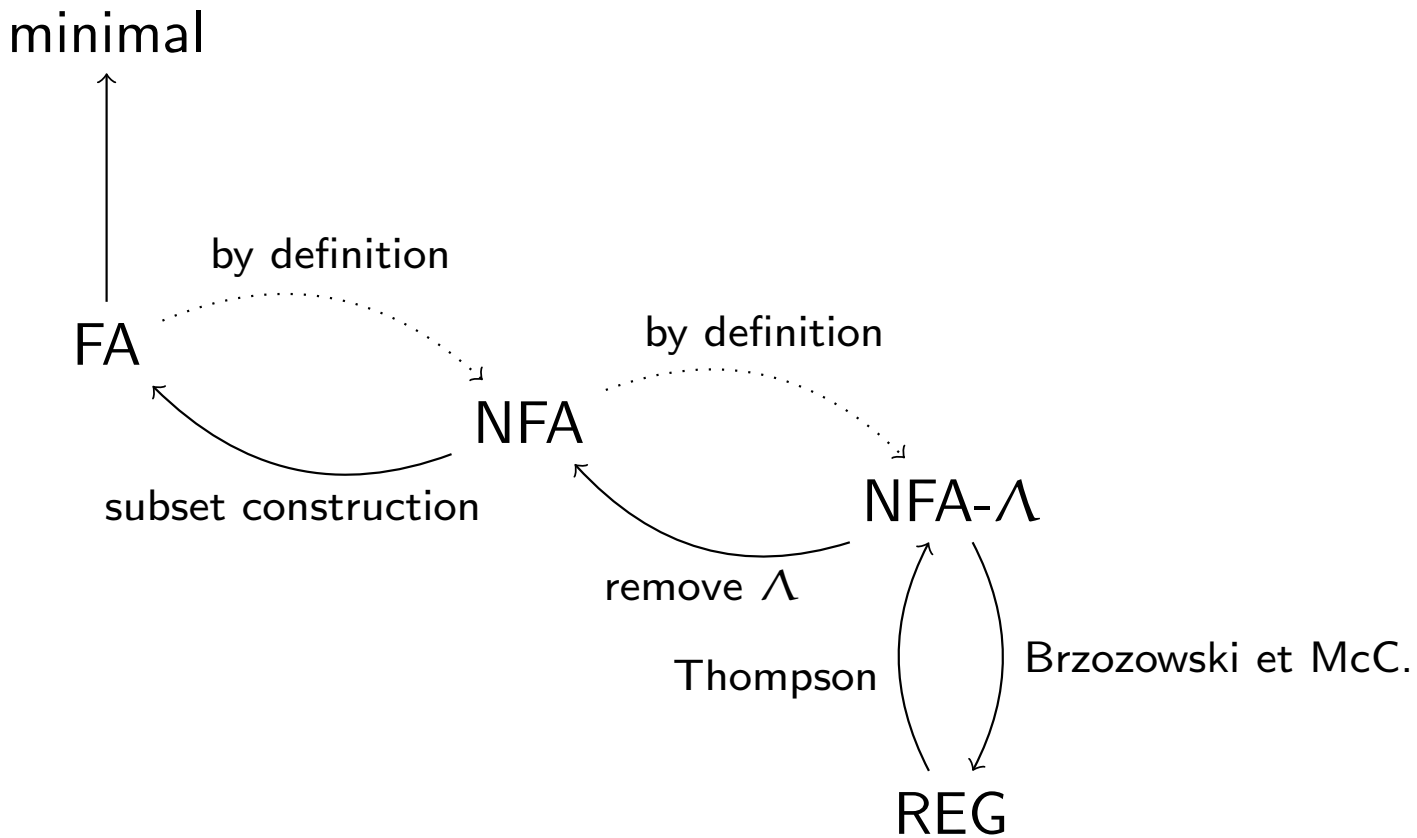
Mark van den Bergh / Rudy van Vliet

Bachelor Informatica
Data Science and Artificial Intelligence
Universiteit Leiden

Fall 2024

Universiteit
Leiden
Leiden Institute of
Advanced Computer Science

# Overview Ch.2 & 3



**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 $L_1^*$.

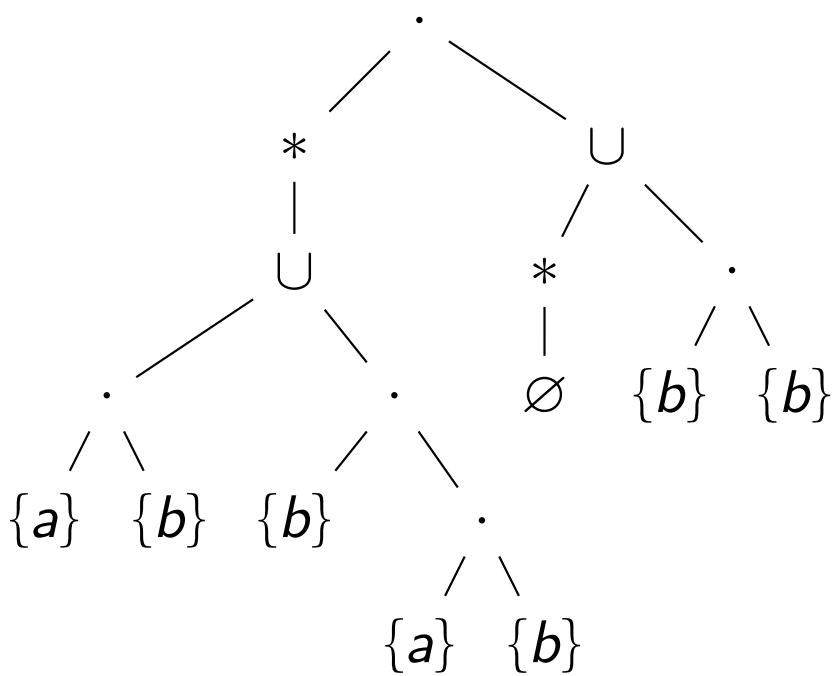
[M] D. 3.1 $\mathcal{R}$

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

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

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



– $\varnothing$, $\Lambda$, 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 |
|---|---|
| $\varnothing$ | $\varnothing$ |
| $\Lambda$ | $\{\Lambda\}$ |
| $a$ | $\{a\}$ |
| $(E_1 + E_2)$ | $L(E_1) \cup L(E_2)$ |
| $(E_1 E_2)$ | $L(E_1) \cdot L(E_2)$ |
| $E_1^*$ | $L(E_1)^*$ |

we say

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

– Odd number of $a$

$bba_0ba_1bbba_2bba_1a_2bb$

Possible RegEx:

$b^*ab^*(ab^*ab^*)^*$

$b^*a(b^*ab^*a)^*b^*$

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

[M] E 3.2

– Ending with $b$, no $aa$

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

$(b + ab)^*(b + ab)$

[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

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

[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

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

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

Use $p$ for $\text{'.'}$

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

[M] E 3.5

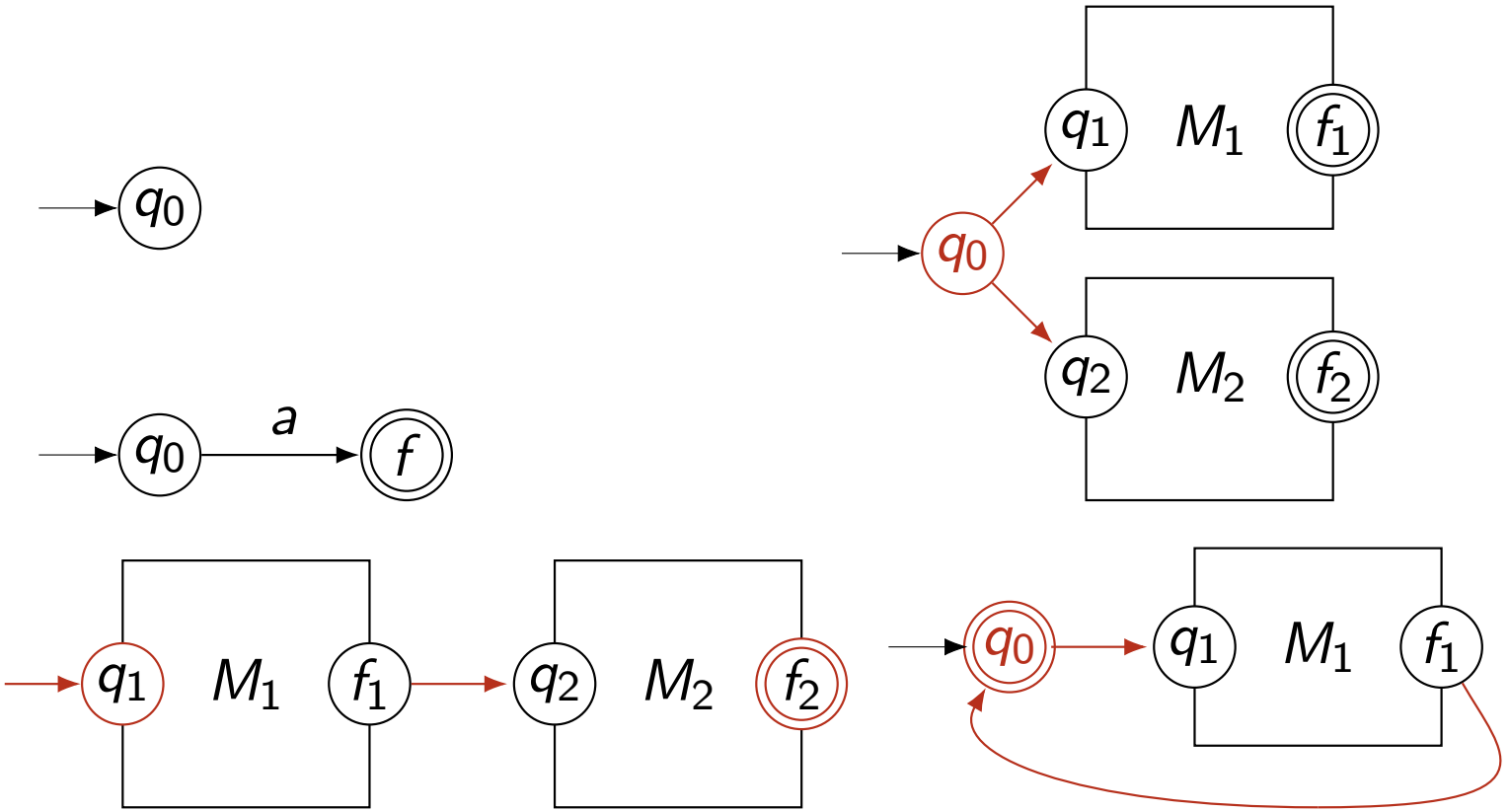
**Theorem (Kleene)**

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

- from RegEx to FA
  $\hookrightarrow$Thompson's construction
- from FA to RegEx
  In book: $\hookrightarrow$McNaughton and Yamada
  We do: $\hookrightarrow$Brzozowski et McCluskey

**Theorem**

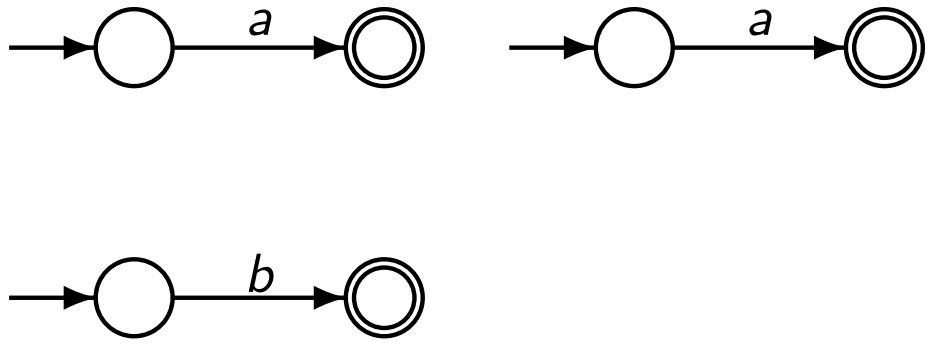
*If $L$ is a regular language, then there exists an NFA that accepts $L$.*

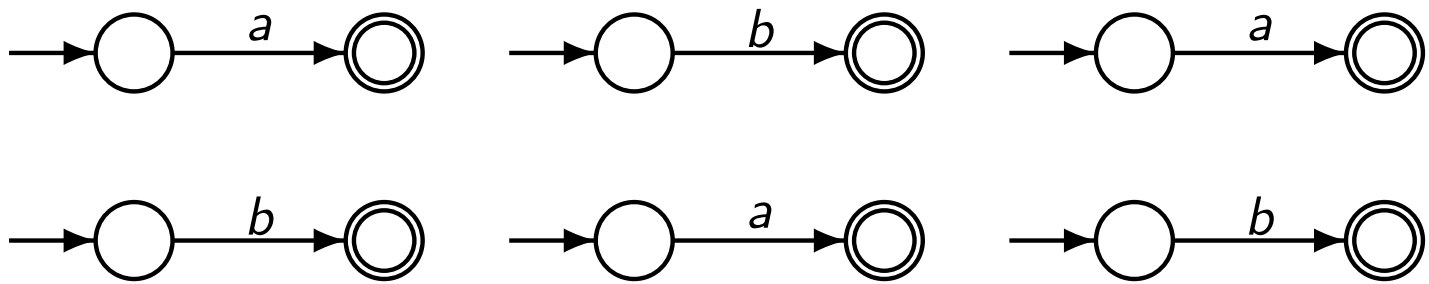



[M] Th 3.25 [L] Th 3.1

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

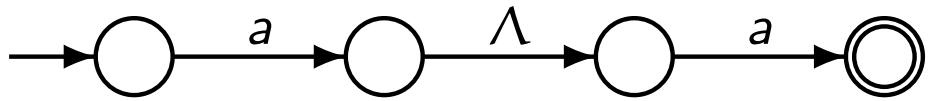
Step 1

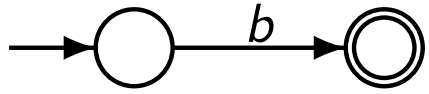


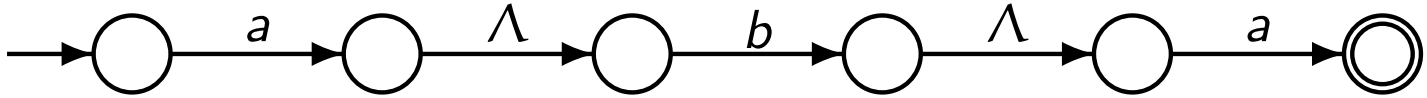



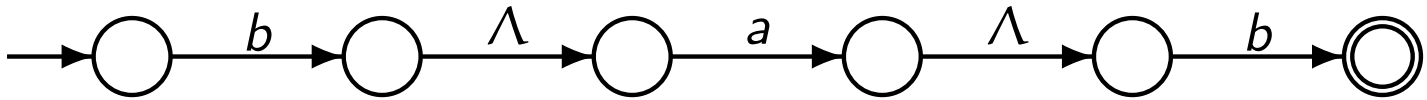
**Example 3.28.** An NFA Corresponding to $((aa + b)^*(aba)^*bab)^*$

Step 2



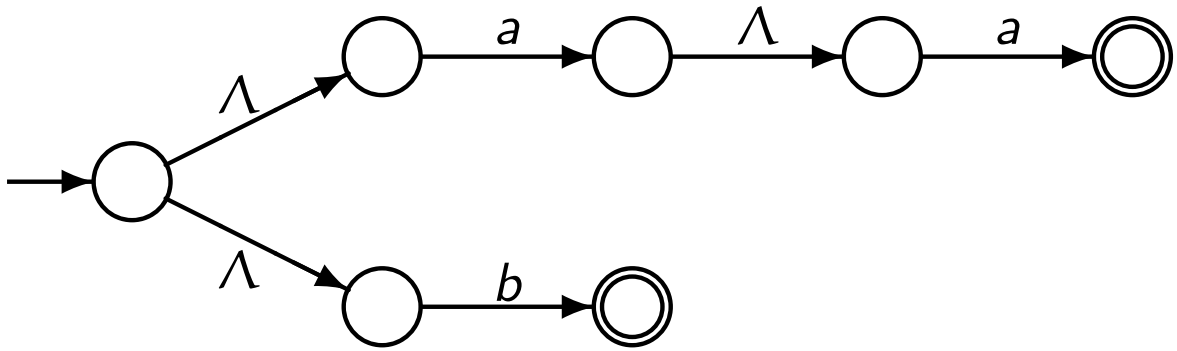



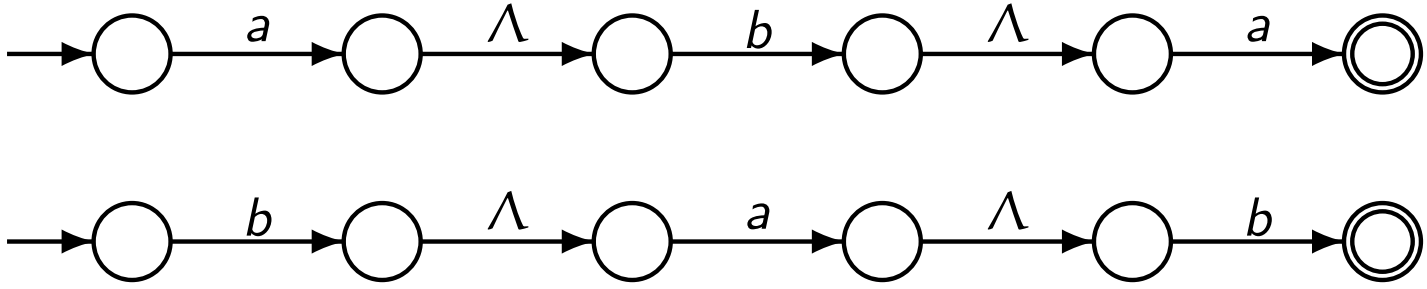



**Example 3.28.** An NFA Corresponding to $((aa + b)^*(aba)^*bab)^*$

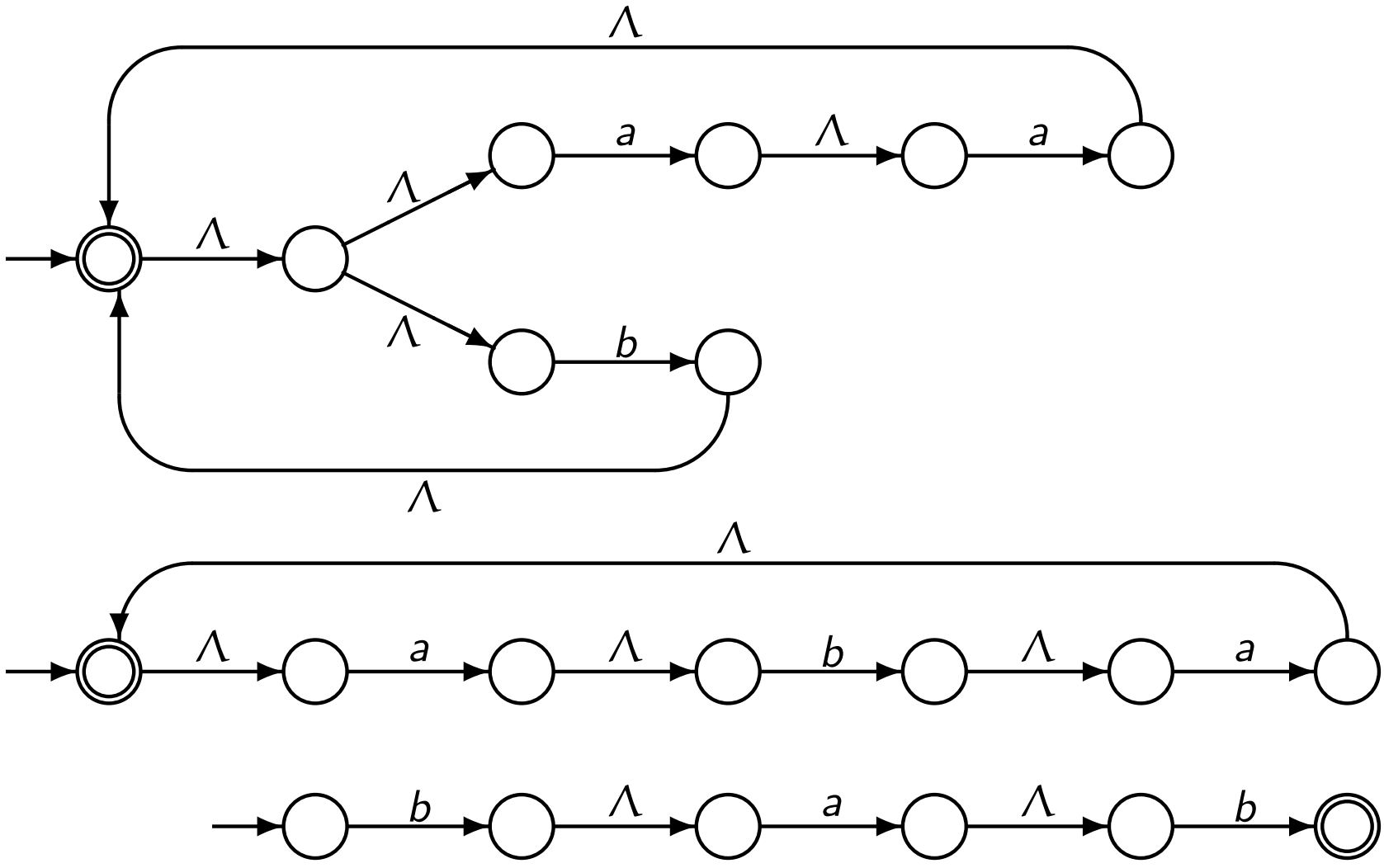
Step 3



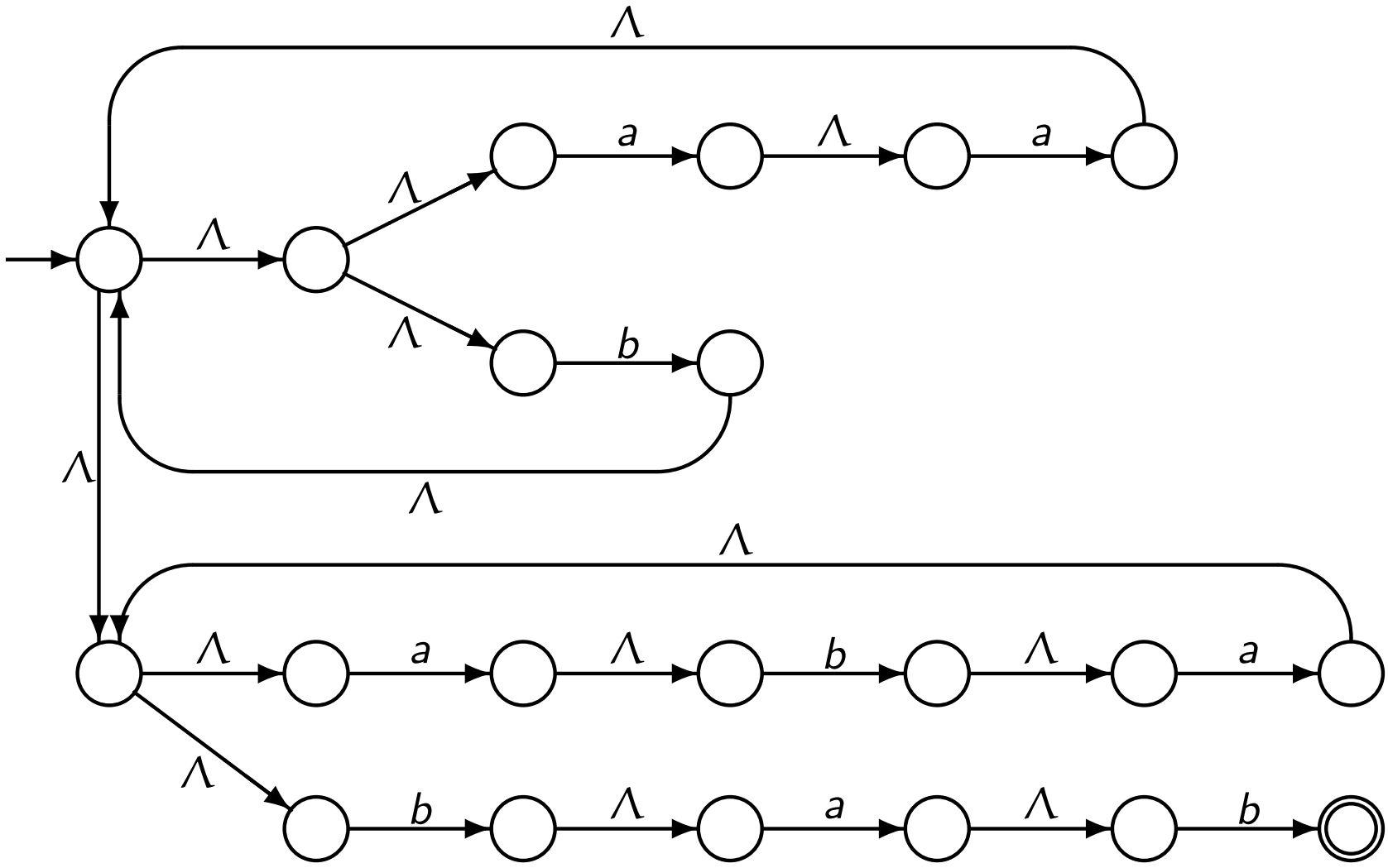
**Example 3.28.** An NFA Corresponding to $((aa + b)^*(aba)^*bab)^*$

Step 4



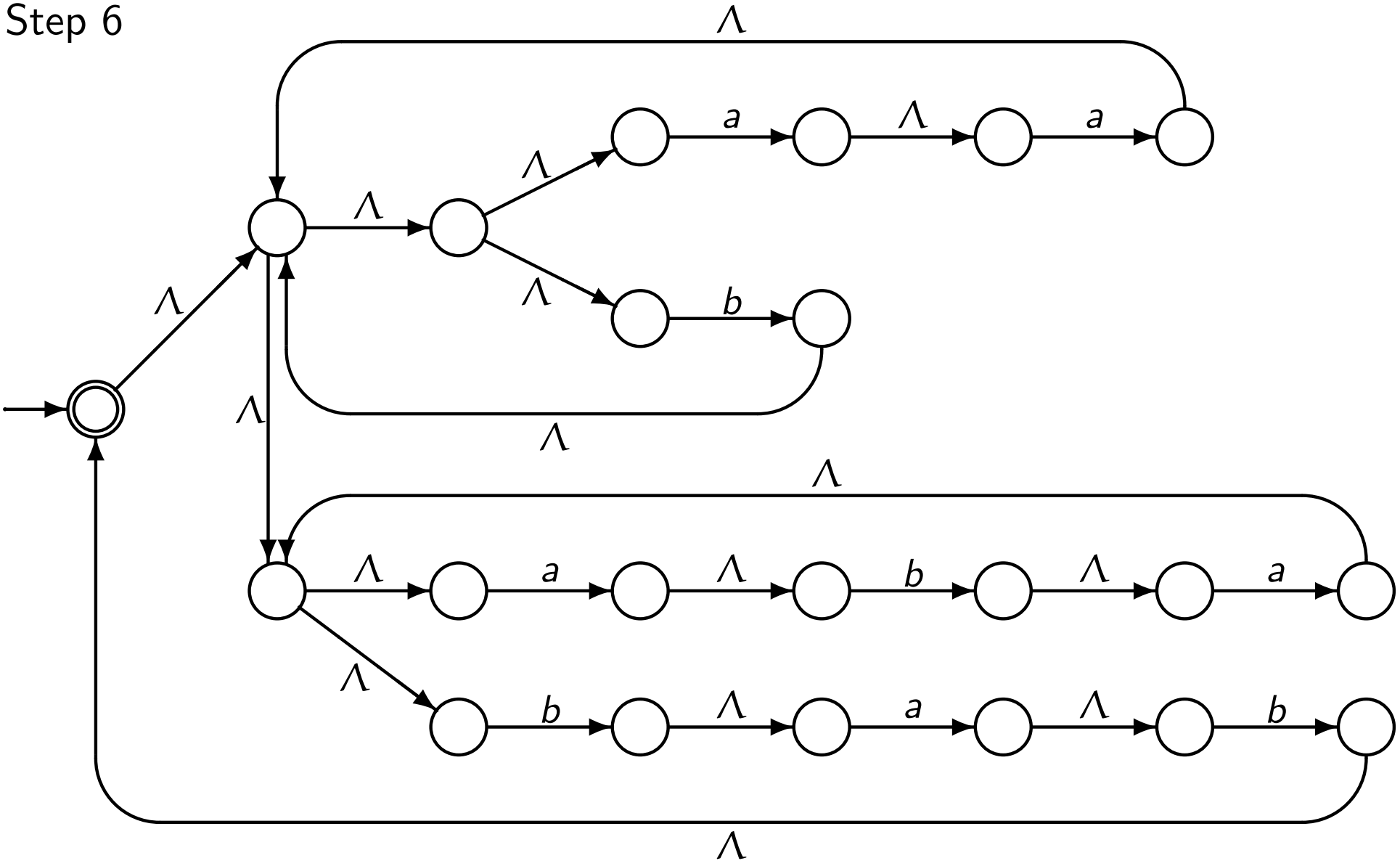
**Example 3.28.** An NFA Corresponding to $((aa + b)^*(aba)^*bab)^*$

Step 5

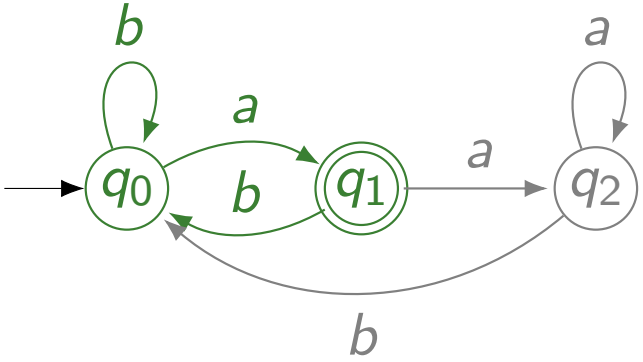


**Example 3.28.** An NFA Corresponding to $((aa + b)^*(aba)^*bab)^*$

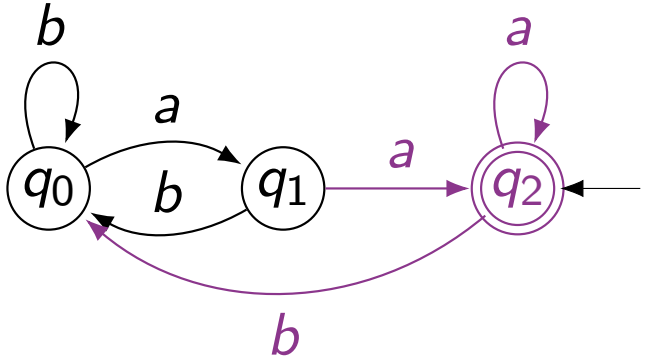
Step 6



# Intro: finding a regular expression



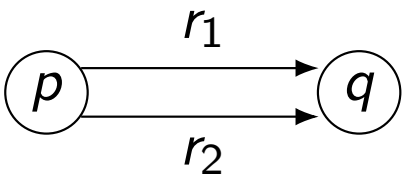

$\underbrace{(b + ab)^* a}_{\text{loop on } q_0}$

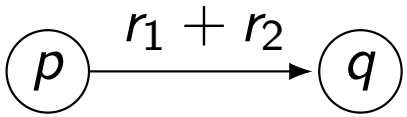



$\underbrace{b\,[(b + ab)^* a]\,a + a}_{\text{single loop on } q_2}$

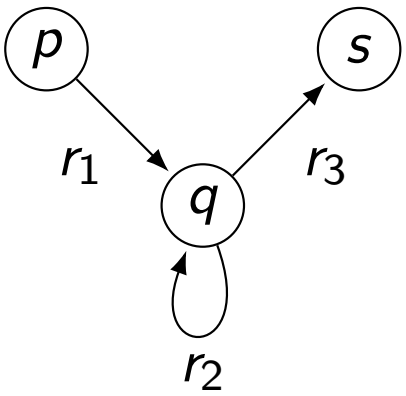
$\underbrace{[(b + ab)^* aa]}_{\text{from } q_0 \text{ to } q_2}\,\underbrace{[b(b + ab)^* aa + a]^*}_{\text{loop on } q_2}$

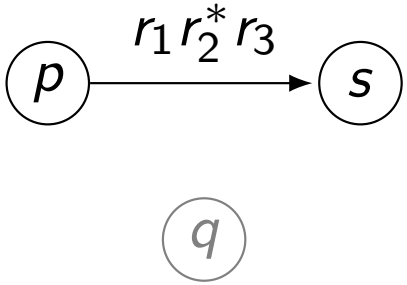
short answer $(a + b)^* aa$ see $\hookrightarrow$FA example



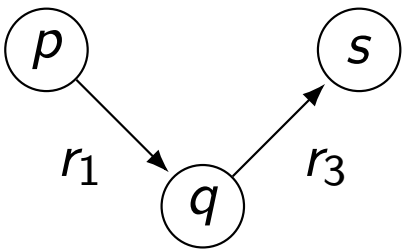



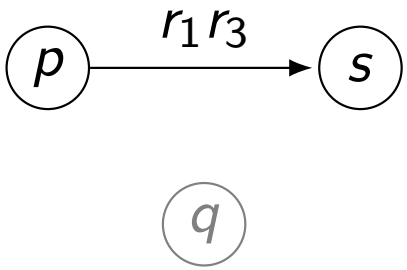

join parallel edges



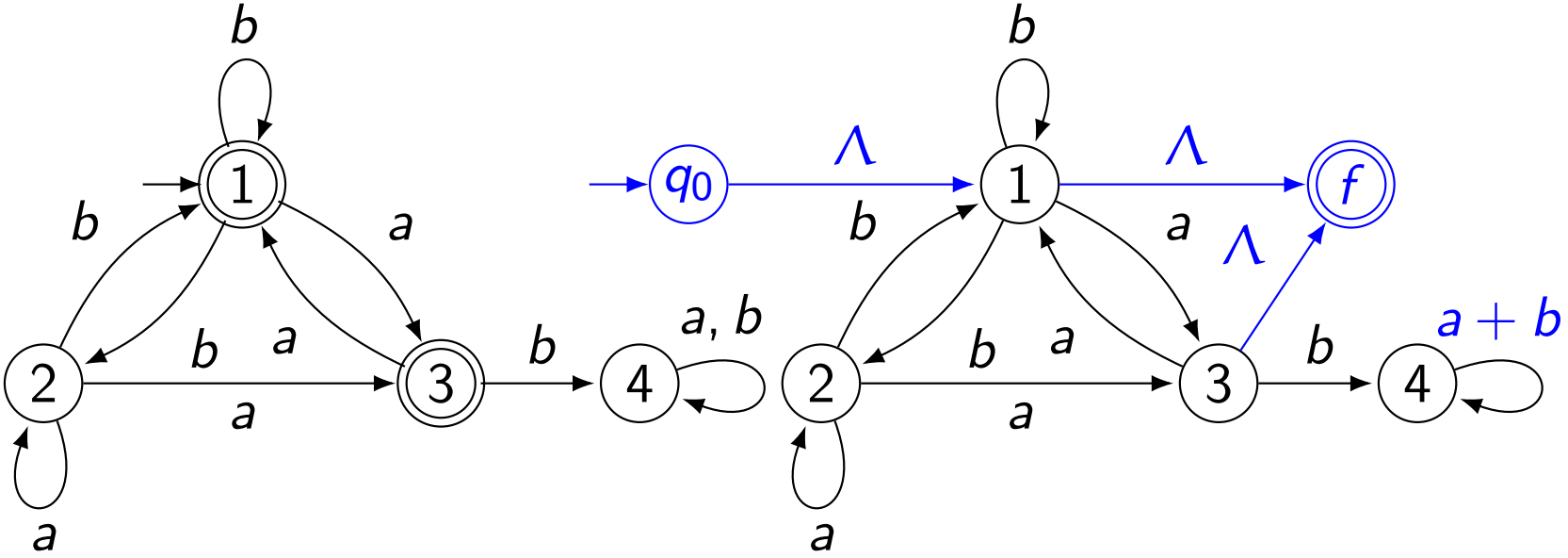

reduce node $q$

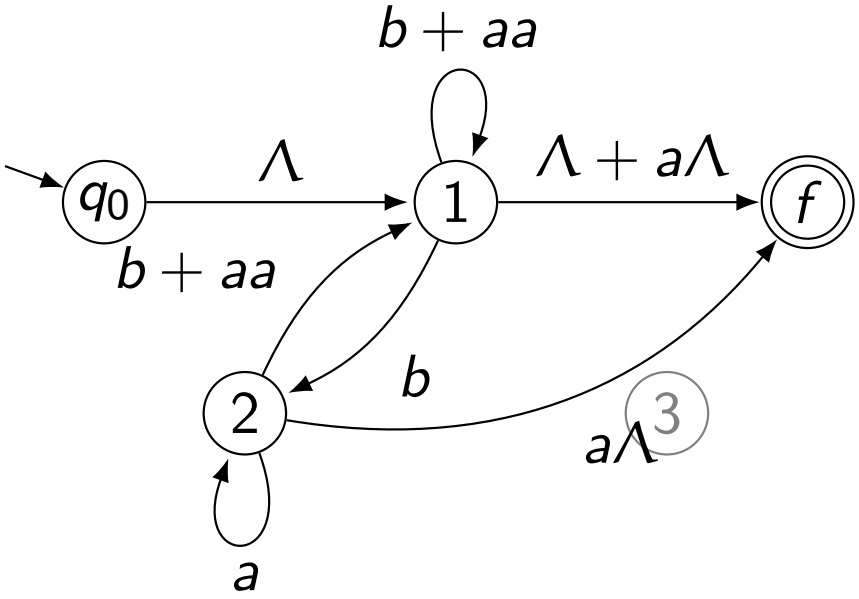


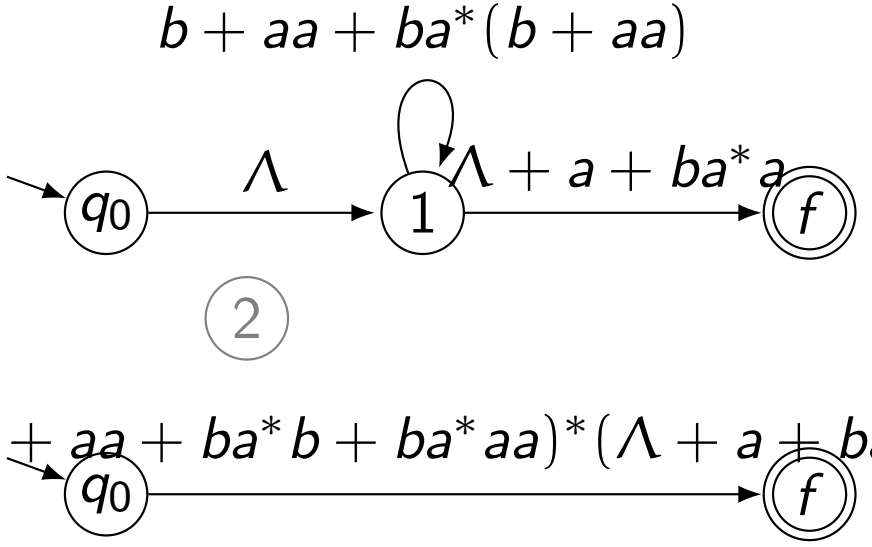




special case: $r_2 = \varnothing$

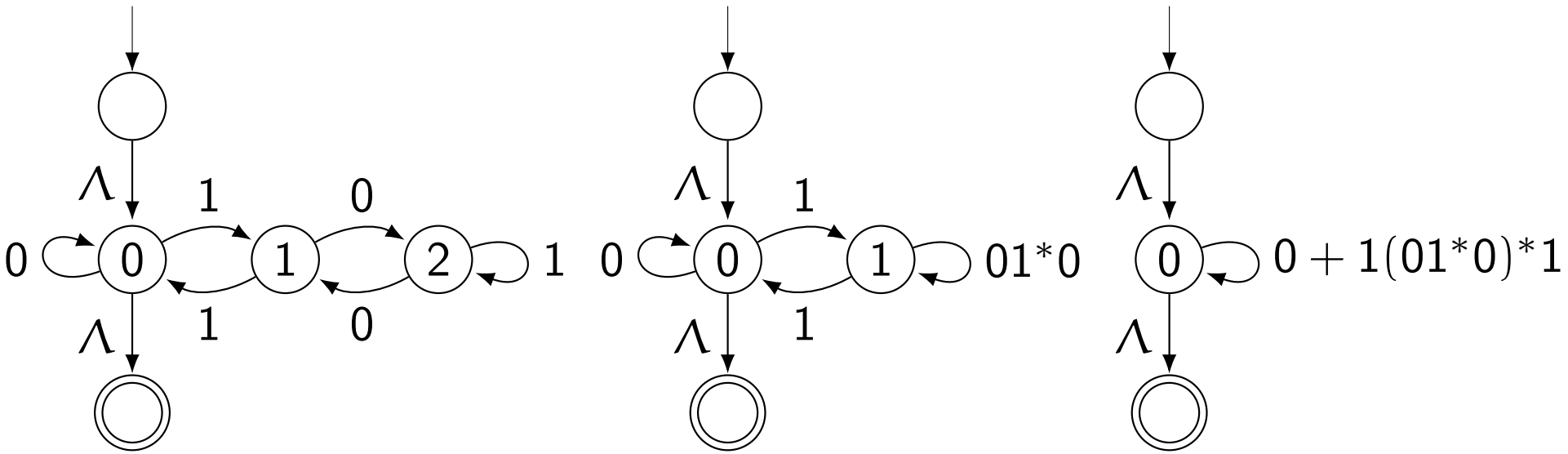
[M] Exercise 3.54


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

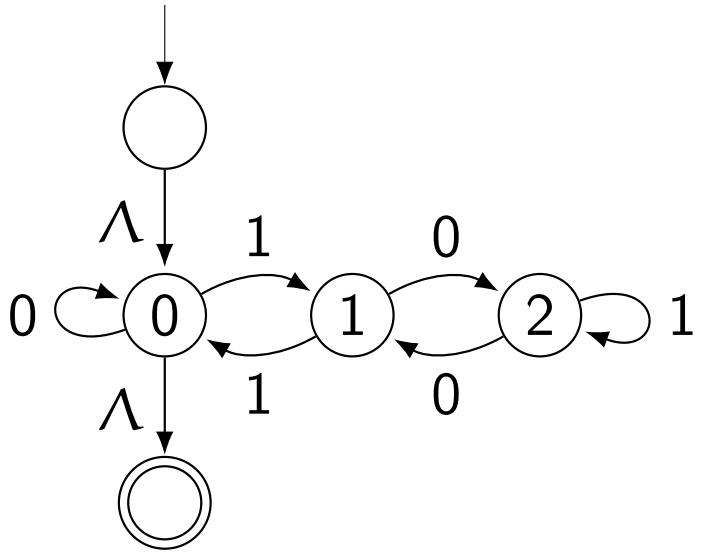

b + aa
q0
Λ
1
Λ + aΛ
f
b + aa
2
b
aΛ
3
a

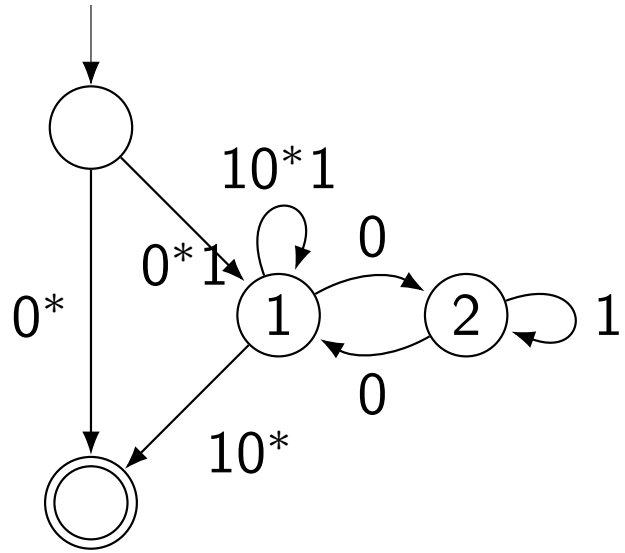

b + aa + ba*(b + aa)
q0
Λ
1
Λ + a + ba*a
f
2
(b + aa + ba*b + ba*aa)*(Λ + a + ba*a)
q0
f

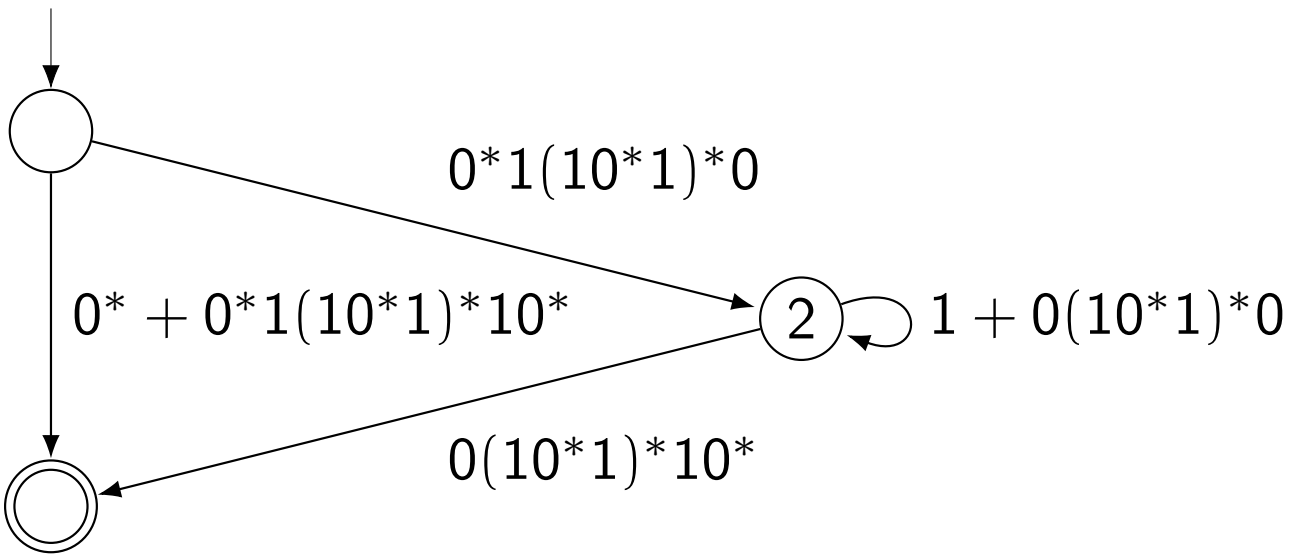
# Example divisible by 3, order 1



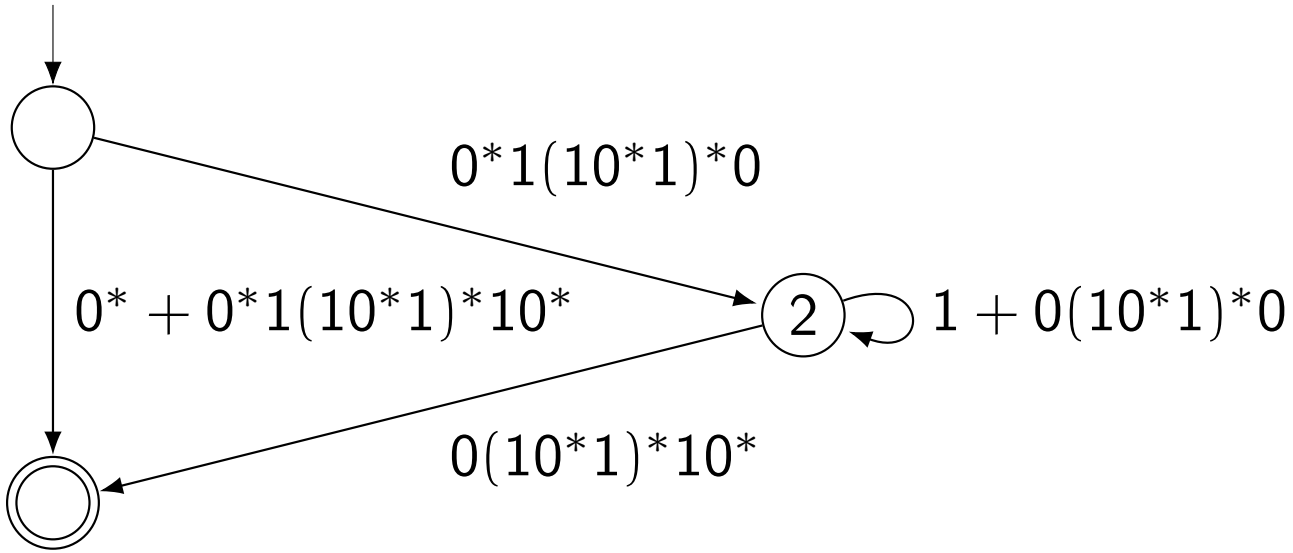
# Example divisible by 3, order 2

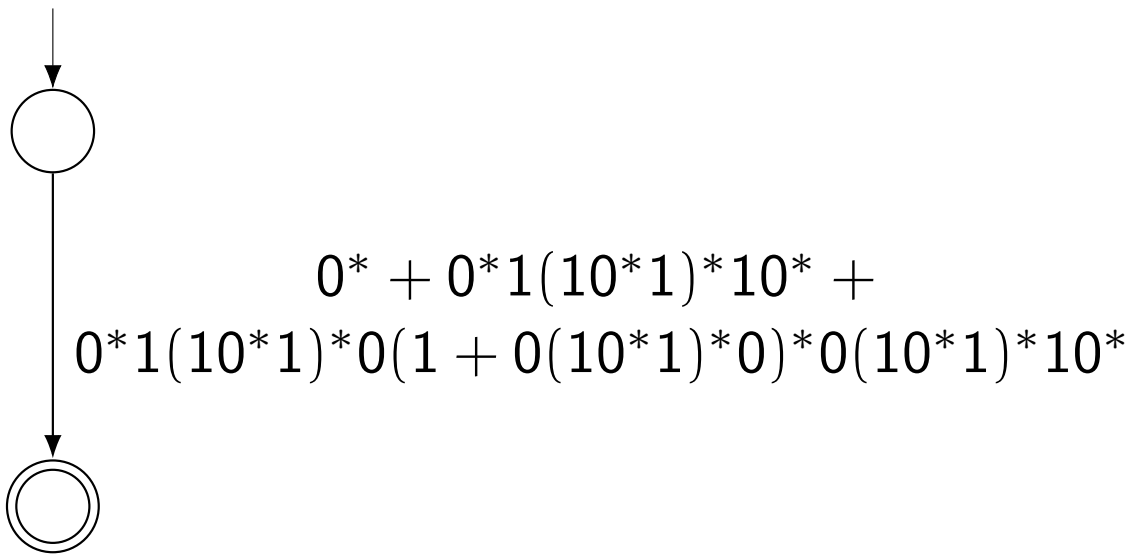


# Example divisible by 3, order 2



# Homomorphism (cf. Ex. 3.53)

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

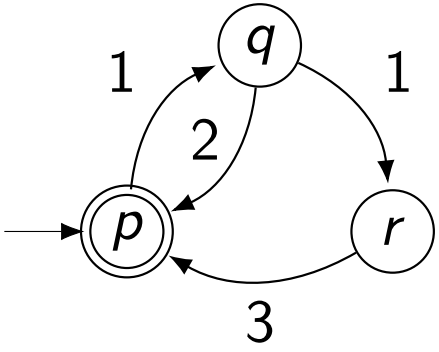
$$h : \begin{array}{rcl} 1 & \mapsto & aa \\ 2 & \mapsto & \Lambda \\ 3 & \mapsto & abb \end{array}$$

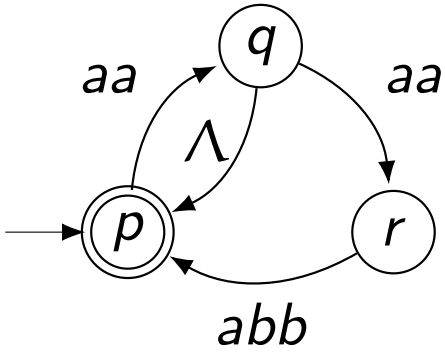
$h : \Sigma_1^* \to \Sigma_2^*$ string-to-string map

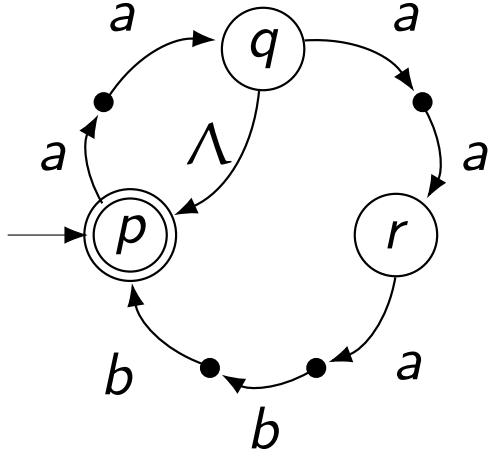
$h(\sigma_1\sigma_2 \ldots \sigma_k) = h(\sigma_1)h(\sigma_2) \ldots h(\sigma_k)$ $\quad h(12113) = aa \cdot \Lambda \cdot aa \cdot aa \cdot abb$

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

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



# Closure

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
- ...