

CS 383

Lecture 04 – Regular Expressions

Stephen Checkoway

Fall, 2023

Review from last time

NFA $N = (Q, \Sigma, \delta, q_0, F)$ where $\delta : Q \times \Sigma_\epsilon \rightarrow P(Q)$ maps a state and an alphabet symbol (or ϵ) to a set of states

We run an NFA on an input w by keeping track of *all* possible states the NFA could be in

We can convert an NFA to a DFA by letting each state of the DFA represent a set of states in the NFA

Building new languages using regular operation

Use regular operations to build new languages

$$A = \{w \mid w \text{ starts and ends with the same symbols}\}$$

$$B = \{b^k a \mid k \geq 1\}$$

$$C = \{\varepsilon, ba, aaa\}$$

$$D = C^*$$

$$E = A \cup (B \circ C)$$

$$F = (D \circ C) \cup (B^* \circ E)$$

Describing complex languages using simpler ones

Use regular operations to break complex languages down into simpler ones

$$A = \{w \mid w \text{ starts and ends with the same symbols}\}$$

$$B = \{b^k a \mid k \geq 1\}$$

$$C = \{\varepsilon, ba, aaa\}$$

Describing complex languages using simpler ones

Use regular operations to break complex languages down into simpler ones

$$A = \{w \mid w \text{ starts and ends with the same symbols}\}$$

$$B = \{b^k a \mid k \geq 1\}$$

$$C = \{\varepsilon, ba, aaa\}$$

$$C = \{\varepsilon\} \cup \{ba\} \cup \{aaa\}$$

Describing complex languages using simpler ones

Use regular operations to break complex languages down into simpler ones

$$A = \{w \mid w \text{ starts and ends with the same symbols}\}$$

$$B = \{b^k a \mid k \geq 1\}$$

$$C = \{\varepsilon, ba, aaa\}$$

$$\begin{aligned} C &= \{\varepsilon\} \cup \{ba\} \cup \{aaa\} \\ &= \{\varepsilon\} \cup (\{b\} \circ \{a\}) \cup (\{a\} \circ \{a\} \circ \{a\}) \end{aligned}$$

Describing complex languages using simpler ones

Use regular operations to break complex languages down into simpler ones

$$A = \{w \mid w \text{ starts and ends with the same symbols}\}$$

$$B = \{b^k a \mid k \geq 1\}$$

$$C = \{\varepsilon, ba, aaa\}$$

$$\begin{aligned} C &= \{\varepsilon\} \cup \{ba\} \cup \{aaa\} \\ &= \{\varepsilon\} \cup (\{b\} \circ \{a\}) \cup (\{a\} \circ \{a\} \circ \{a\}) \end{aligned}$$

$$B = \{b\} \circ \{b\}^* \circ \{a\}$$

Describing complex languages using simpler ones

Use regular operations to break complex languages down into simpler ones

$$A = \{w \mid w \text{ starts and ends with the same symbols}\}$$

$$B = \{b^k a \mid k \geq 1\}$$

$$C = \{\varepsilon, ba, aaa\}$$

$$\begin{aligned} C &= \{\varepsilon\} \cup \{ba\} \cup \{aaa\} \\ &= \{\varepsilon\} \cup (\{b\} \circ \{a\}) \cup (\{a\} \circ \{a\} \circ \{a\}) \end{aligned}$$

$$B = \{b\} \circ \{b\}^* \circ \{a\}$$

$$A = \{a\} \cup \{b\} \cup (\{a\} \circ \Sigma^* \circ \{a\}) \cup (\{b\} \circ \Sigma^* \circ \{b\})$$

Describing complex languages using simpler ones

Use regular operations to break complex languages down into simpler ones

$$A = \{w \mid w \text{ starts and ends with the same symbols}\}$$

$$B = \{b^k a \mid k \geq 1\}$$

$$C = \{\varepsilon, ba, aaa\}$$

$$\begin{aligned} C &= \{\varepsilon\} \cup \{ba\} \cup \{aaa\} \\ &= \{\varepsilon\} \cup (\{b\} \circ \{a\}) \cup (\{a\} \circ \{a\} \circ \{a\}) \end{aligned}$$

$$B = \{b\} \circ \{b\}^* \circ \{a\}$$

$$\begin{aligned} A &= \{a\} \cup \{b\} \cup (\{a\} \circ \Sigma^* \circ \{a\}) \cup (\{b\} \circ \Sigma^* \circ \{b\}) \\ &= \{a\} \cup \{b\} \cup (\{a\} \circ (\{a\} \cup \{b\})^* \circ \{a\}) \cup (\{b\} \circ (\{a\} \cup \{b\})^* \circ \{b\}) \end{aligned}$$

Describing complex languages using simpler ones

Use regular operations to break complex languages down into simpler ones

$$A = \{w \mid w \text{ starts and ends with the same symbols}\}$$

$$B = \{b^k a \mid k \geq 1\}$$

$$C = \{\varepsilon, ba, aaa\}$$

$$\begin{aligned} C &= \{\varepsilon\} \cup \{ba\} \cup \{aaa\} \\ &= \{\varepsilon\} \cup (\{b\} \circ \{a\}) \cup (\{a\} \circ \{a\} \circ \{a\}) \end{aligned}$$

$$B = \{b\} \circ \{b\}^* \circ \{a\}$$

$$\begin{aligned} A &= \{a\} \cup \{b\} \cup (\{a\} \circ \Sigma^* \circ \{a\}) \cup (\{b\} \circ \Sigma^* \circ \{b\}) \\ &= \{a\} \cup \{b\} \cup (\{a\} \circ (\{a\} \cup \{b\})^* \circ \{a\}) \cup (\{b\} \circ (\{a\} \cup \{b\})^* \circ \{b\}) \end{aligned}$$

We broke each language down into languages containing $\{a\}$, $\{b\}$, or $\{\varepsilon\}$ and combined them using the three regular operations \cup , \circ , and *

Regular expressions

The braces aren't adding anything since all of our sets are singletons; let's drop them

Similarly, let's drop the \circ much as how we drop multiplication symbols

Let's also replace \cup with $|$ (which we read as "or")

This gives us **regular expressions** (regex)

$$A = \{a\} \cup \{b\} \cup (\{a\} \circ (\{a\} \cup \{b\})^* \circ \{a\}) \cup (\{b\} \circ (\{a\} \cup \{b\})^* \circ \{b\})$$

=

$$B = \{b\} \circ \{b\}^* \circ \{a\}$$

=

$$C = \{\varepsilon\} \cup (\{b\} \circ \{a\}) \cup (\{a\} \circ \{a\} \circ \{a\})$$

=

Regular expressions

The braces aren't adding anything since all of our sets are singletons; let's drop them
Similarly, let's drop the \circ much as how we drop multiplication symbols
Let's also replace \cup with $|$ (which we read as "or")

This gives us **regular expressions** (regex)

$$\begin{aligned} A &= \{a\} \cup \{b\} \cup (\{a\} \circ (\{a\} \cup \{b\})^* \circ \{a\}) \cup (\{b\} \circ (\{a\} \cup \{b\})^* \circ \{b\}) \\ &= \underline{a \mid b \mid a(a \mid b)^* a \mid b(a \mid b)^* b} \end{aligned}$$

$$B = \{b\} \circ \{b\}^* \circ \{a\}$$

=

$$C = \{\varepsilon\} \cup (\{b\} \circ \{a\}) \cup (\{a\} \circ \{a\} \circ \{a\})$$

=

Regular expressions

The braces aren't adding anything since all of our sets are singletons; let's drop them
Similarly, let's drop the \circ much as how we drop multiplication symbols
Let's also replace \cup with $|$ (which we read as "or")

This gives us **regular expressions** (regex)

$$\begin{aligned} A &= \{a\} \cup \{b\} \cup (\{a\} \circ (\{a\} \cup \{b\})^* \circ \{a\}) \cup (\{b\} \circ (\{a\} \cup \{b\})^* \circ \{b\}) \\ &= \underline{a \mid b \mid a(a \mid b)^* a \mid b(a \mid b)^* b} \end{aligned}$$

$$\begin{aligned} B &= \{b\} \circ \{b\}^* \circ \{a\} \\ &= \underline{bb^* a} \end{aligned}$$

$$\begin{aligned} C &= \{\varepsilon\} \cup (\{b\} \circ \{a\}) \cup (\{a\} \circ \{a\} \circ \{a\}) \\ &= \end{aligned}$$

Regular expressions

The braces aren't adding anything since all of our sets are singletons; let's drop them
Similarly, let's drop the \circ much as how we drop multiplication symbols
Let's also replace \cup with $|$ (which we read as "or")

This gives us **regular expressions** (regex)

$$\begin{aligned} A &= \{a\} \cup \{b\} \cup (\{a\} \circ (\{a\} \cup \{b\})^* \circ \{a\}) \cup (\{b\} \circ (\{a\} \cup \{b\})^* \circ \{b\}) \\ &= \underline{a \mid b \mid a(a \mid b)^* a \mid b(a \mid b)^* b} \end{aligned}$$

$$\begin{aligned} B &= \{b\} \circ \{b\}^* \circ \{a\} \\ &= \underline{bb^* a} \end{aligned}$$

$$\begin{aligned} C &= \{\varepsilon\} \cup (\{b\} \circ \{a\}) \cup (\{a\} \circ \{a\} \circ \{a\}) \\ &= \underline{\varepsilon \mid ba \mid aaa} \end{aligned}$$

Regular expressions

The braces aren't adding anything since all of our sets are singletons; let's drop them
Similarly, let's drop the \circ much as how we drop multiplication symbols
Let's also replace \cup with $|$ (which we read as "or")

This gives us **regular expressions** (regex)

$$\begin{aligned} A &= \{a\} \cup \{b\} \cup (\{a\} \circ (\{a\} \cup \{b\})^* \circ \{a\}) \cup (\{b\} \circ (\{a\} \cup \{b\})^* \circ \{b\}) \\ &= \underline{a \mid b \mid a(a \mid b)^* a \mid b(a \mid b)^* b} \end{aligned}$$

$$\begin{aligned} B &= \{b\} \circ \{b\}^* \circ \{a\} \\ &= \underline{bb^* a} \end{aligned}$$

$$\begin{aligned} C &= \{\varepsilon\} \cup (\{b\} \circ \{a\}) \cup (\{a\} \circ \{a\} \circ \{a\}) \\ &= \underline{\varepsilon \mid ba \mid aaa} \end{aligned}$$

Order of operation: * , \circ , $|$

Parentheses used for grouping

We underline the expression to differentiate the string `aaa` from the regular expression `aaa`

Regular expressions

Six types of regular expressions: three base types, three recursive types

Regex	Language	
\emptyset	\emptyset	(very rarely used)
ε	$\{\varepsilon\}$	
t	$\{t\}$	for each $t \in \Sigma$
$R_1 \mid R_2$	$L(R_1) \cup L(R_2)$	R_1 and R_2 are regex
$R_1 \circ R_2$	$L(R_1) \circ L(R_2)$	R_1 and R_2 are regex
R^*	$L(R)^*$	R is a regex

As a shorthand, we'll use $\underline{\Sigma}$ to mean $\underline{a \mid b}$ (or similar for other alphabets)

$$A = \underline{a \mid b \mid a\Sigma^*a \mid b\Sigma^*b}$$

Technicalities

Technically, a regular expression **generates** or **describes** a (regular) language, it is not a language itself

Given a regular expression R , the language $L(R)$ is the set of strings generated by R

E.g., $R = \underline{ab^*a}$ generates strings aa , aba , $abba$, \dots

$$L(R) = \{ab^k a \mid k \geq 0\}$$

A DFA M **recognizes** a (regular) language $L(M)$ but we don't identify M with its language

Similarly, we shouldn't identify a regular expression R with its language $L(R)$; however it is customary to do so

Still, even if we let $\{aba\} = \underline{aba}$, that doesn't mean aba is the same as \underline{aba} !

Kleene star

- $\underline{a}^* = \{a^k \mid k \geq 0\}$

Kleene star

- $\underline{a}^* = \{a^k \mid k \geq 0\}$
- $\underline{(a \mid b \mid c)}^* = \{w \mid w \text{ contains any number of } a, b, \text{ or } c \text{ in any order}\}$

Kleene star

- $\underline{a}^* = \{a^k \mid k \geq 0\}$
- $\underline{(a \mid b \mid c)}^* = \{w \mid w \text{ contains any number of } a, b, \text{ or } c \text{ in any order}\}$
- $\underline{(aa \mid bab)}^* = \{w \mid w \text{ is the concatenation of 0 or more } aa \text{ or } bab\}$

Kleene star

- $\underline{a^*} = \{a^k \mid k \geq 0\}$
- $\underline{(a \mid b \mid c)^*} = \{w \mid w \text{ contains any number of } a, b, \text{ or } c \text{ in any order}\}$
- $\underline{(aa \mid bab)^*} = \{w \mid w \text{ is the concatenation of 0 or more } aa \text{ or } bab\}$
- $\underline{a^*b^*} = \{a^m b^n \mid m, n \geq 0\}$

Kleene star

- $\underline{a}^* = \{a^k \mid k \geq 0\}$
- $\underline{(a \mid b \mid c)}^* = \{w \mid w \text{ contains any number of } a, b, \text{ or } c \text{ in any order}\}$
- $\underline{(aa \mid bab)}^* = \{w \mid w \text{ is the concatenation of 0 or more } aa \text{ or } bab\}$
- $\underline{a^*b^*} = \{a^m b^n \mid m, n \geq 0\}$
- $\underline{\varepsilon}^* = \{\varepsilon\} = \underline{\varepsilon}$

Kleene star

- $\underline{a}^* = \{a^k \mid k \geq 0\}$
- $\underline{(a \mid b \mid c)}^* = \{w \mid w \text{ contains any number of } a, b, \text{ or } c \text{ in any order}\}$
- $\underline{(aa \mid bab)}^* = \{w \mid w \text{ is the concatenation of 0 or more } aa \text{ or } bab\}$
- $\underline{a^*b^*} = \{a^m b^n \mid m, n \geq 0\}$
- $\underline{\varepsilon}^* = \{\varepsilon\} = \underline{\varepsilon}$
- $\underline{\emptyset}^* = \{\varepsilon\} = \underline{\varepsilon}$

Kleene star

- $\underline{a}^* = \{a^k \mid k \geq 0\}$
- $\underline{(a \mid b \mid c)}^* = \{w \mid w \text{ contains any number of } a, b, \text{ or } c \text{ in any order}\}$
- $\underline{(aa \mid bab)}^* = \{w \mid w \text{ is the concatenation of 0 or more } aa \text{ or } bab\}$
- $\underline{a^*b^*} = \{a^m b^n \mid m, n \geq 0\}$
- $\underline{\varepsilon}^* = \{\varepsilon\} = \underline{\varepsilon}$
- $\underline{\emptyset}^* = \{\varepsilon\} = \underline{\varepsilon}$
- $\underline{\Sigma}^* = \Sigma^* = \{w \mid w \text{ is a string over } \Sigma\}$

Regular expression examples

- $\underline{\Sigma\Sigma} = \{w \mid |w| = 2\}$

Regular expression examples

- $\underline{\Sigma\Sigma} = \{w \mid |w| = 2\}$
- $\underline{(\Sigma\Sigma)^*} = \{w \mid |w| \text{ is even}\}$

Regular expression examples

- $\underline{\Sigma\Sigma} = \{w \mid |w| = 2\}$
- $\underline{(\Sigma\Sigma)^*} = \{w \mid |w| \text{ is even}\}$
- $\underline{a^*(baa^*)^*} = \{w \mid \text{every } b \text{ in } w \text{ is followed by at least one } a\}$

Regular expression examples

- $\underline{\Sigma\Sigma} = \{w \mid |w| = 2\}$
- $\underline{(\Sigma\Sigma)^*} = \{w \mid |w| \text{ is even}\}$
- $\underline{a^*(baa^*)^*} = \{w \mid \text{every } b \text{ in } w \text{ is followed by at least one } a\}$
- $\underline{(a \mid \varepsilon)b^*} = \underline{ab^* \mid b^*}$

Regular expression examples

- $\underline{\Sigma\Sigma} = \{w \mid |w| = 2\}$
- $\underline{(\Sigma\Sigma)^*} = \{w \mid |w| \text{ is even}\}$
- $\underline{a^*(baa^*)^*} = \{w \mid \text{every } b \text{ in } w \text{ is followed by at least one } a\}$
- $\underline{(a \mid \varepsilon)b^*} = \underline{ab^* \mid b^*}$
- $\underline{a^*ba^*} = \{w \mid w \text{ contains exactly one } b\}$

Question 1

What strings are in the language given by the regular expression $(a \mid bb)(\epsilon \mid a)$?

Question 1

What strings are in the language given by the regular expression $(a \mid bb)(\epsilon \mid a)$?

a, aa, bb, bba

Question 2

True or false. If languages A and B each contain 2 strings, then $A \circ B$ contains 4 strings.

Question 2

True or false. If languages A and B each contain 2 strings, then $A \circ B$ contains 4 strings.

False. Counter example: $A = B = \{\varepsilon, a\}$. $A \circ B = \{\varepsilon, a, aa\}$

Another counter example $A = \{a, ab\}$ and $B = \{b, bb\}$. $A \circ B = \{ab, abb, abbb\}$

Question 3

Is abaaa in the language given by $(a \mid ba \mid aaa)^*$?

Question 3

Is abaaa in the language given by $(a \mid ba \mid aaa)^*$?

Yes. $abaaa = a ba a a$

Question 4

Write a regex for the language $\{w \mid \text{baba is a substring of } w\}$

Question 4

Write a regex for the language $\{w \mid \text{baba is a substring of } w\}$

$\Sigma^* \text{baba} \Sigma^*$

Question 5

Write a regex for the language

$\{w \mid \text{the second symbol of } w \text{ is a or the third to last symbol of } w \text{ is b}\}$

Question 5

Write a regex for the language

$\{w \mid \text{the second symbol of } w \text{ is } a \text{ or the third to last symbol of } w \text{ is } b\}$

$\Sigma a \Sigma^* \mid \Sigma^* b \Sigma \Sigma$

Question 6

Let $\Sigma = \{0, 1, \dots, 9, -\}$ and $D = \underline{0 \mid 1 \mid \dots \mid 9}$. What strings are generated by the following regular expression?

$$\underline{((1 - \mid \varepsilon)DDD - \mid \varepsilon)DDD - DDD}$$

Question 6

Let $\Sigma = \{0, 1, \dots, 9, -\}$ and $D = \underline{0 \mid 1 \mid \dots \mid 9}$. What strings are generated by the following regular expression?

$$\underline{((1 - \mid \varepsilon)DDD - \mid \varepsilon)DDD - DDD}$$

U.S. phone numbers.

We can rewrite this regex as

$$\underline{1-DDD-DDD-DDDD \mid DDD-DDD-DDDD \mid DDD-DDDD}$$

Question 7

If R is a regular expression, then the language generated by $\underline{R^*}$ is either infinite or contains exactly one string. Under what condition on R is $\underline{R^*}$ infinite? When $\underline{R^*}$ contains exactly one string, what is the string and what is R ?

Question 7

If R is a regular expression, then the language generated by $\underline{R^*}$ is either infinite or contains exactly one string. Under what condition on R is $\underline{R^*}$ infinite? When $\underline{R^*}$ contains exactly one string, what is the string and what is R ?

$\underline{R^*}$ is infinite if R contains at least one nonempty string

$\underline{R^*}$ contains exactly one string, namely ε , when $R = \underline{\varepsilon}$ or $R = \underline{\emptyset}$

Regular expression manipulation

Let R_1 , R_2 , and R_3 be regular expressions

- $R_1 \mid \emptyset$ = R_1

Regular expression manipulation

Let R_1 , R_2 , and R_3 be regular expressions

- $\underline{R_1} \mid \emptyset = R_1$
- $\underline{R_1 \circ \varepsilon} = R_1$

Regular expression manipulation

Let R_1 , R_2 , and R_3 be regular expressions

- $\underline{R_1 \mid \emptyset} = R_1$
- $\underline{R_1 \circ \varepsilon} = R_1$
- $\underline{(R_1 \mid R_2)R_3} = \underline{R_1R_3 \mid R_2R_3}$

Regular expression manipulation

Let R_1 , R_2 , and R_3 be regular expressions

- $\underline{R_1 \mid \emptyset} = R_1$
- $\underline{R_1 \circ \varepsilon} = R_1$
- $\underline{(R_1 \mid R_2)R_3} = \underline{R_1 R_3 \mid R_2 R_3}$
- $\underline{R_1(R_2 \mid R_3)} = \underline{R_1 R_2 \mid R_1 R_3}$

Regular expression manipulation

Let R_1 , R_2 , and R_3 be regular expressions

- $\underline{R_1 \mid \emptyset} = R_1$
- $\underline{R_1 \circ \varepsilon} = R_1$
- $\underline{(R_1 \mid R_2)R_3} = \underline{R_1R_3 \mid R_2R_3}$
- $\underline{R_1(R_2 \mid R_3)} = \underline{R_1R_2 \mid R_1R_3}$
- $\underline{(R_1^*)^*} = \underline{R_1^*}$

Regular expression manipulation

Let R_1 , R_2 , and R_3 be regular expressions

- $\underline{R_1 \mid \emptyset} = R_1$
- $\underline{R_1 \circ \varepsilon} = R_1$
- $\underline{(R_1 \mid R_2)R_3} = \underline{R_1 R_3 \mid R_2 R_3}$
- $\underline{R_1(R_2 \mid R_3)} = \underline{R_1 R_2 \mid R_1 R_3}$
- $\underline{(R_1^*)^*} = \underline{R_1^*}$
- $\underline{(R_1 \mid R_2)^*} = \underline{(R_1^* R_2^*)^*}$

Regular expression manipulation

Let R_1 , R_2 , and R_3 be regular expressions

- $\underline{R_1 \mid \emptyset} = R_1$
- $\underline{R_1 \circ \varepsilon} = R_1$
- $\underline{(R_1 \mid R_2)R_3} = \underline{R_1R_3 \mid R_2R_3}$
- $\underline{R_1(R_2 \mid R_3)} = \underline{R_1R_2 \mid R_1R_3}$
- $\underline{(R_1^*)^*} = \underline{R_1^*}$
- $\underline{(R_1 \mid R_2)^*} = \underline{(R_1^*R_2^*)^*}$

Theorem

Every regular expression R can be rewritten as an equivalent regular expression

$$\underline{R_1 \mid R_2 \mid \cdots \mid R_k}$$

such that none of the R_i contain an “or” (\mid)

Converting regular expressions to NFAs

Theorem

Every regular expression R can be converted to an equivalent NFA N . I.e., $L(N) = L(R)$

Proof idea

Induction on the structure of the regex

We need to construct NFAs directly for the three base cases, $\underline{\emptyset}$, $\underline{\varepsilon}$ and \underline{t} for $t \in \Sigma$


Then, we handle the three inductive cases, $\underline{R_1 \mid R_2}$, $\underline{R_1 \circ R_2}$, and $\underline{R_1^*}$

For the inductive cases, we assume there exist NFAs for R_1 and R_2 and use them to build NFAs for the three inductive cases

Converting regular expressions to NFAs

Proof.

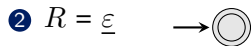
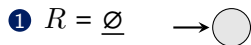
Base cases.

① $R = \underline{\emptyset}$ \rightarrow 

Converting regular expressions to NFAs

Proof.

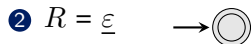
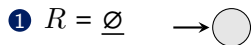
Base cases.



Converting regular expressions to NFAs

Proof.


Base cases.




Converting regular expressions to NFAs

Proof.

Base cases.

① $R = \underline{\emptyset}$ \rightarrow 

② $R = \underline{\varepsilon}$ \rightarrow 

③ $R = \underline{t}$ \rightarrow  for $t \in \Sigma$

Inductive cases.

④ $R = \underline{R_1 \mid R_2}$

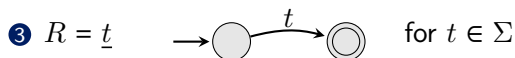
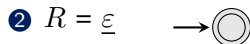
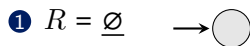
⑤ $R = \underline{R_1 \circ R_2}$

⑥ $R = \underline{R_1^*}$

Converting regular expressions to NFAs

Proof.

Base cases.



Inductive cases.

④ $R = \underline{R_1 \mid R_2}$

⑤ $R = \underline{R_1 \circ R_2}$


⑥ $R = \underline{R_1^*}$


By the inductive hypothesis, there exist NFAs N_1 and N_2 such that $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.

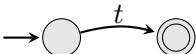
Converting regular expressions to NFAs

Proof.

Base cases.

① $R = \underline{\emptyset}$ \rightarrow 

② $R = \underline{\varepsilon}$ \rightarrow 

③ $R = \underline{t}$ \rightarrow  for $t \in \Sigma$

Inductive cases.

④ $R = \underline{R_1 \mid R_2}$

⑤ $R = \underline{R_1 \circ R_2}$

⑥ $R = \underline{R_1^*}$

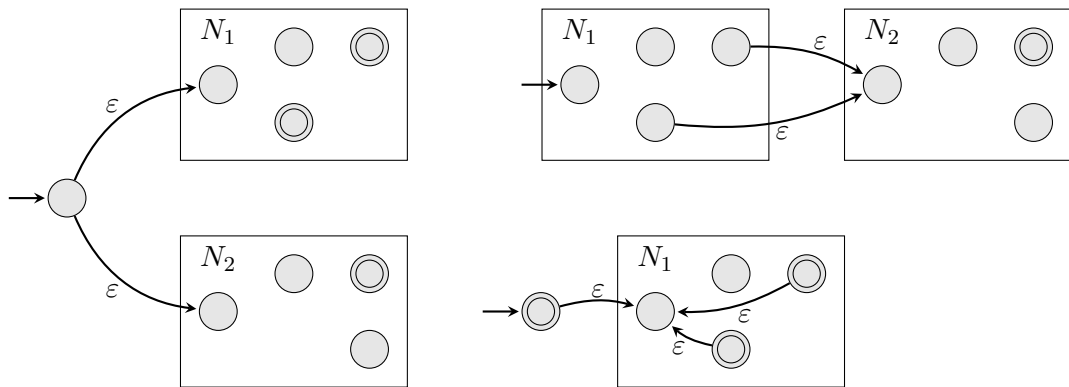
By the inductive hypothesis, there exist NFAs N_1 and N_2 such that $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.

Since regular languages are closed under union, concatenation, and Kleene star, $L(R)$ is regular so there exists some NFA N such that $L(N) = L(R)$. □

Converting regular expressions to NFAs

The proof of the inductive cases applied previous theorems to show that some NFA exists

But we know how to perform the constructions explicitly:



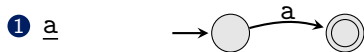
Regular expressions describe regular languages

The language of a regular expression is regular

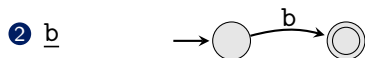
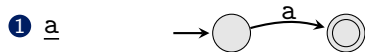
This follows directly from the previous theorem:

Regular expression \Rightarrow NFA \Rightarrow DFA \Rightarrow regular language

Regular expression to NFA: $R = \underline{a(ba)^* \mid b(ab)^*}$

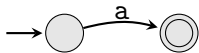


Regular expression to NFA: $R = \underline{a(ba)^* \mid b(ab)^*}$

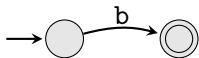


Regular expression to NFA: $R = \underline{a(ba)^* \mid b(ab)^*}$

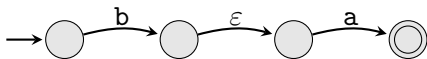
① a



② b

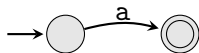


③ ba

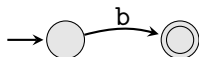


Regular expression to NFA: $R = \underline{a(ba)^* \mid b(ab)^*}$

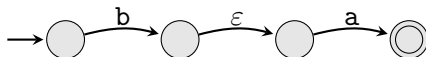
① a



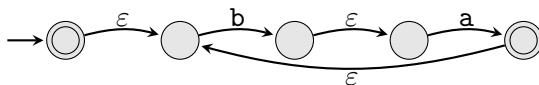
② b



③ ba

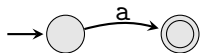


④ $(ba)^*$

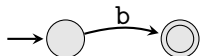


Regular expression to NFA: $R = \underline{a(ba)^* \mid b(ab)^*}$

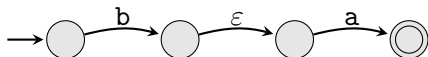
① a



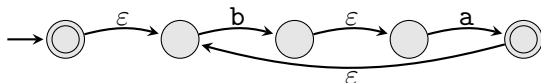
② b



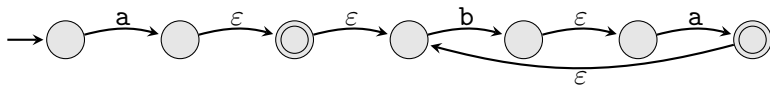
③ ba



④ (ba)*

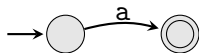


⑤ a(ba)*

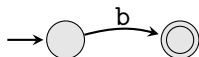


Regular expression to NFA: $R = \underline{a(ba)^* \mid b(ab)^*}$

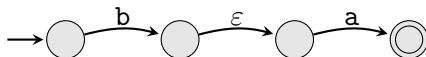
① a



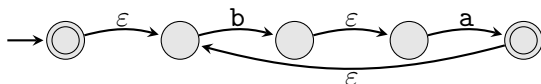
② b



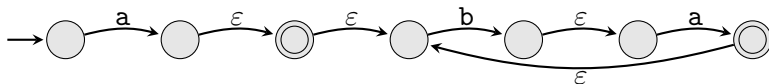
③ ba



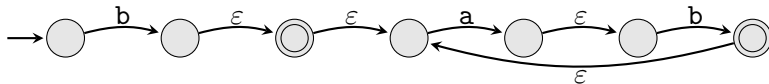
④ (ba)*



⑤ a(ba)*

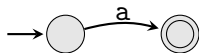


⑥ b(ab)*

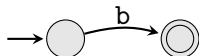


Regular expression to NFA: $R = \underline{a(ba)^* \mid b(ab)^*}$

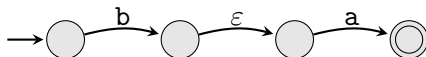
① a



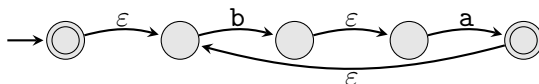
② b



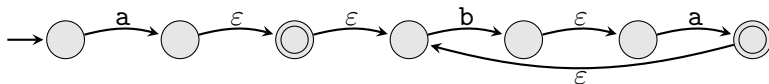
③ ba



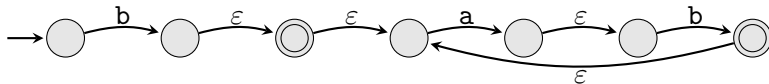
④ (ba)*



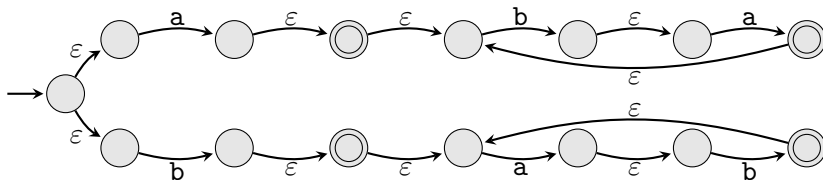
⑤ a(ba)*



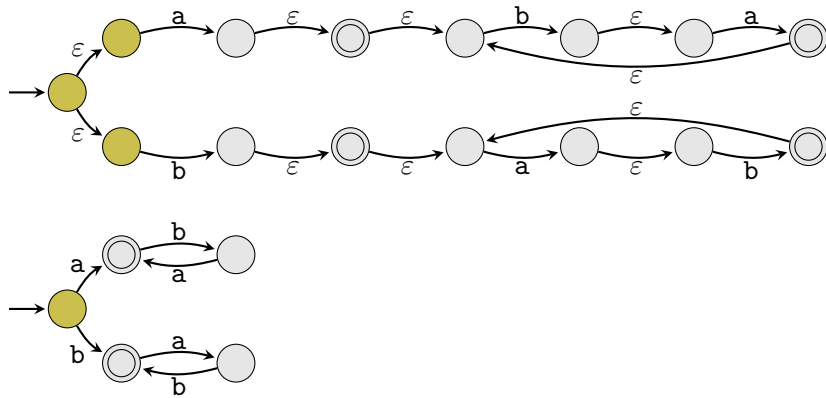
⑥ b(ab)*



⑦ R

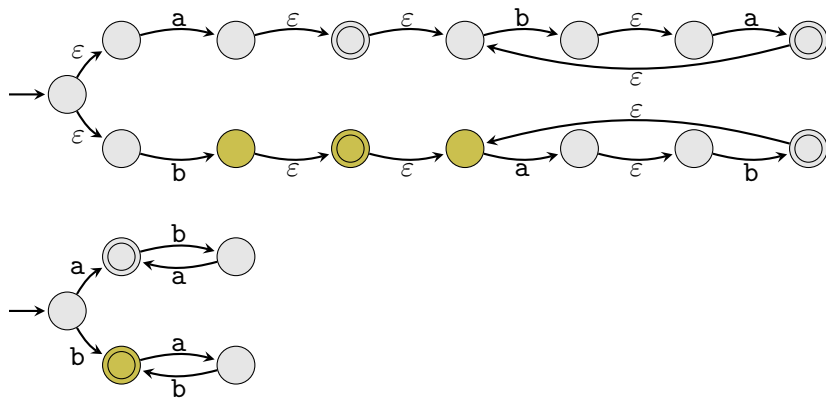


Not the smallest possible NFA



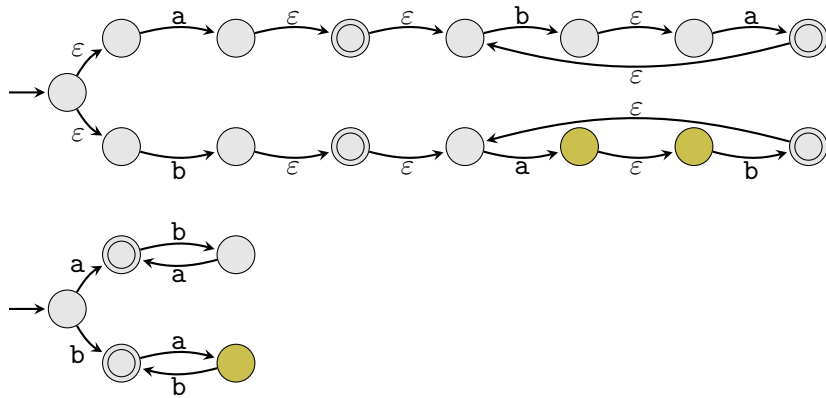
• babab

Not the smallest possible NFA



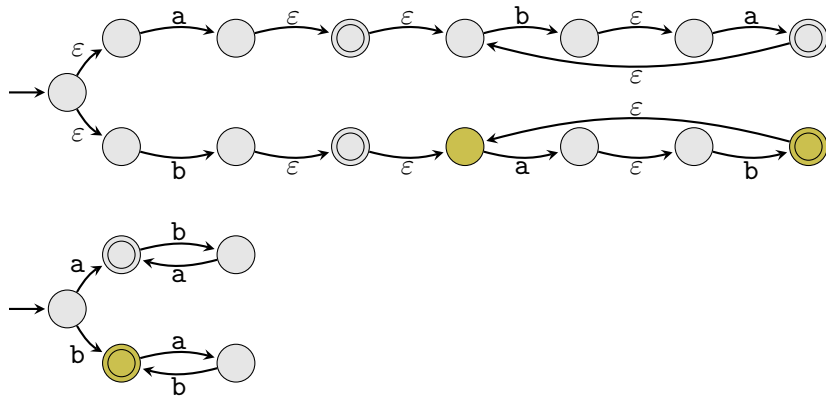
• babab

Not the smallest possible NFA



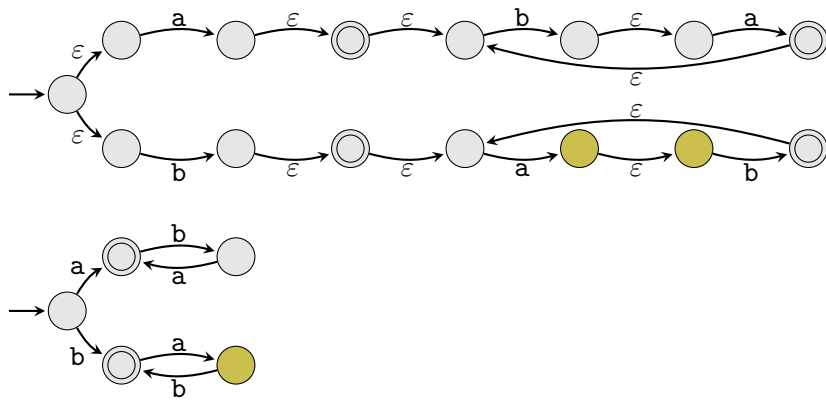
• babab

Not the smallest possible NFA



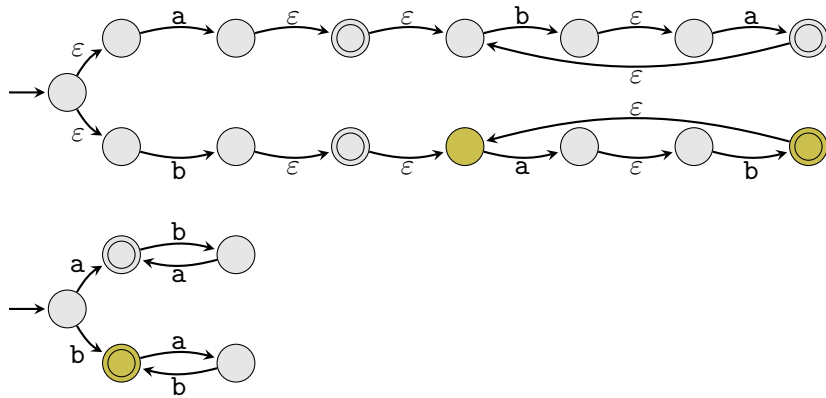
- bab**a**b

Not the smallest possible NFA



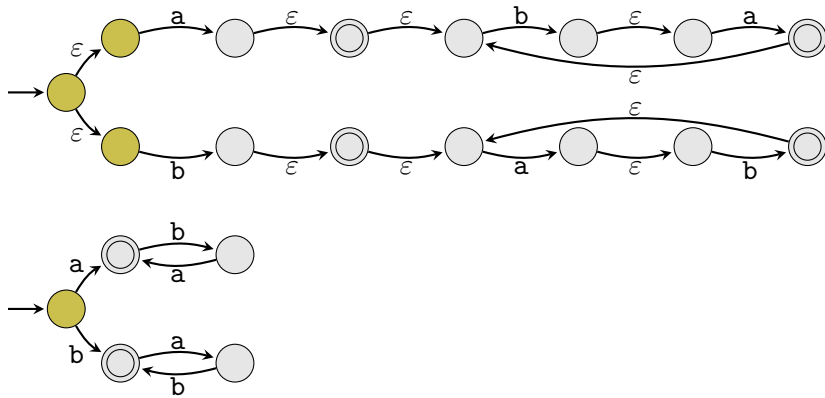
• babab

Not the smallest possible NFA



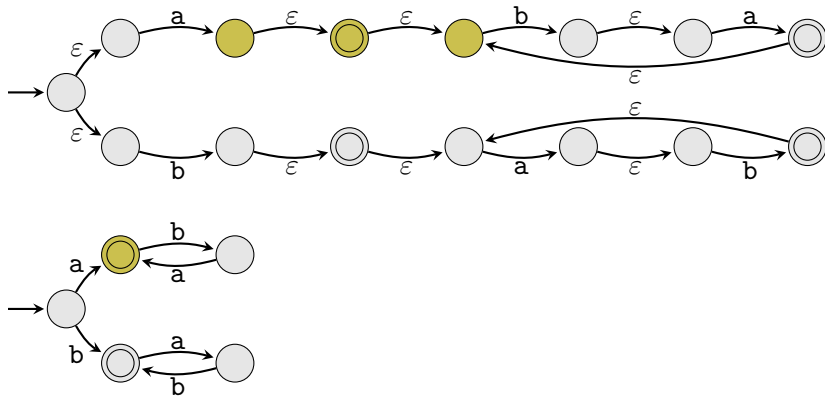
• babab ✓ Accepted

Not the smallest possible NFA



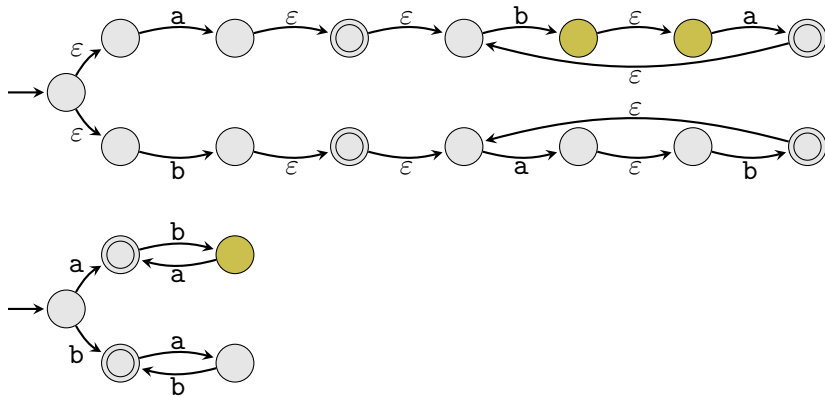
- babab ✓ Accepted
- abab

Not the smallest possible NFA



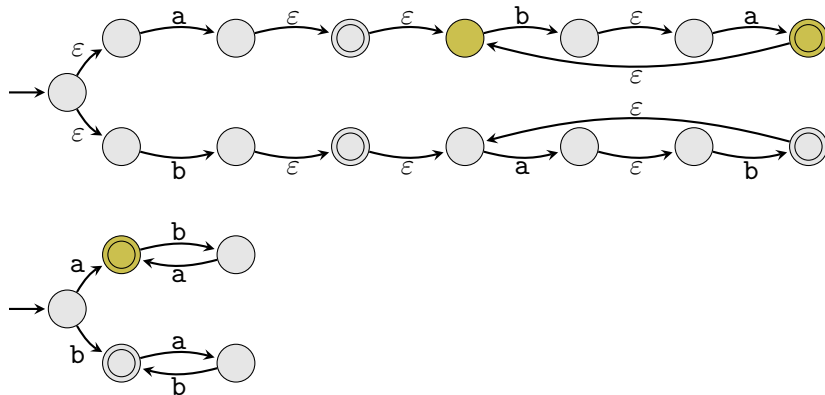
- babab ✓ Accepted
- abab

Not the smallest possible NFA



- babab ✓ Accepted
- abab

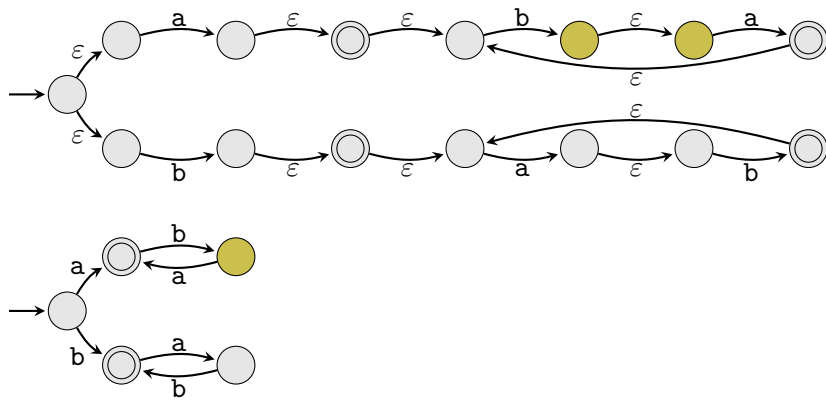
Not the smallest possible NFA



• babab ✓ Accepted

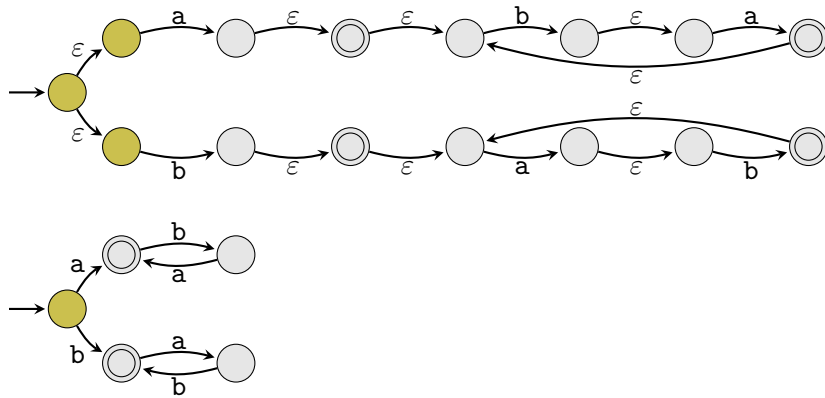
• aba**b**

Not the smallest possible NFA



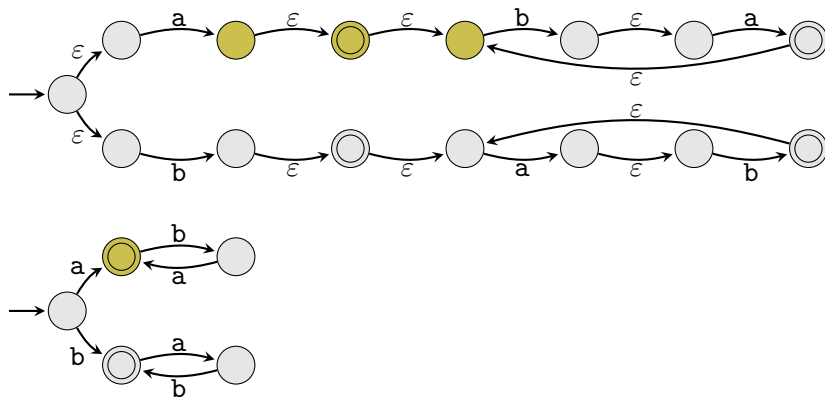
- babab ✓ Accepted
- abab ✗ Rejected

Not the smallest possible NFA



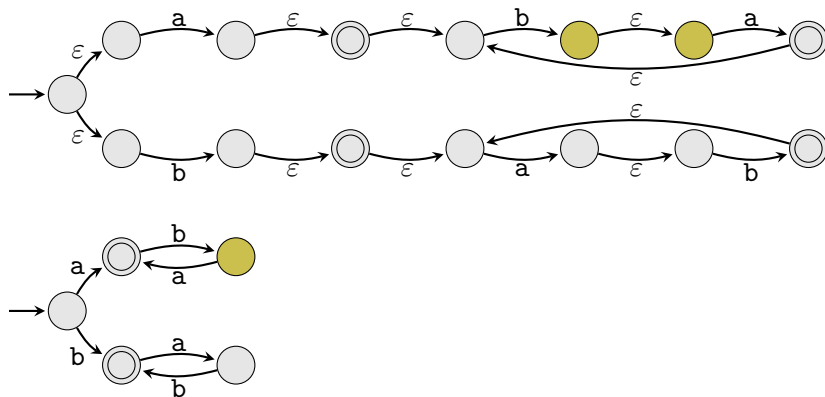
- babab ✓ Accepted
- abab ✗ Rejected
- abb

Not the smallest possible NFA



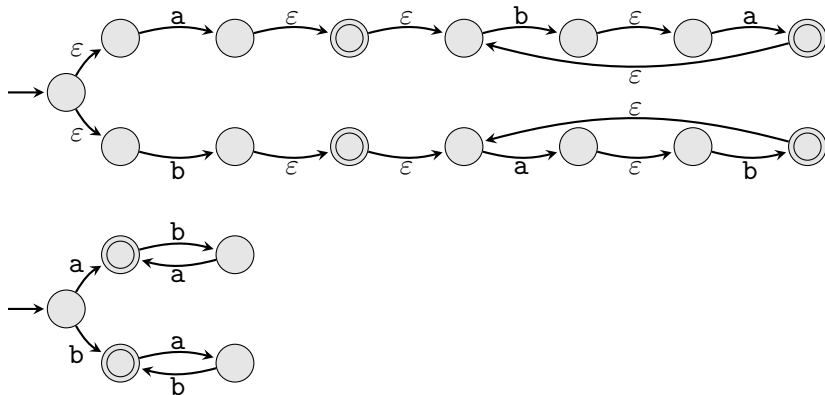
- babab ✓ Accepted
- abab ✗ Rejected
- a**b**

Not the smallest possible NFA



- babab ✓ Accepted
- abab ✗ Rejected
- ab**b**

Not the smallest possible NFA



- babab ✓ Accepted
- abab ✗ Rejected
- abb ✗ Rejected

Converting from NFAs to regex

Theorem

Every NFA (and thus every DFA) can be converted to an equivalent regular expression.

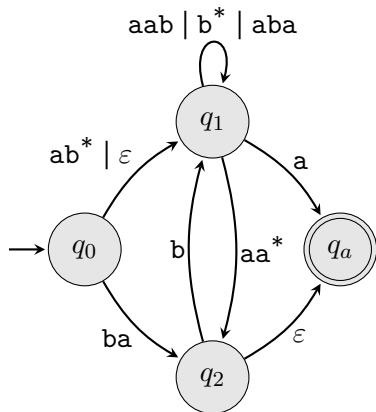
Proof idea

- ① Convert the NFA to a new type of finite automaton whose edges are labeled with regular expressions
- ② Remove states and update transitions one at a time from the new automaton to produce an equivalent automaton
- ③ When only the start and (single) accept state remain, the transition between them is the regular expression

Generalized NFA (GNFA)

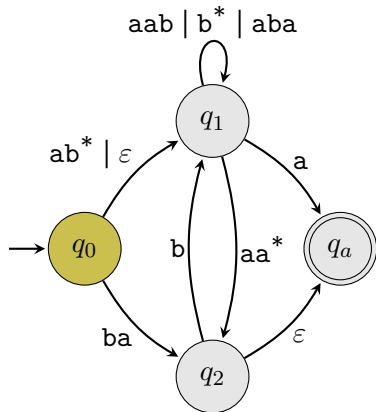
A GNFA is a finite automaton with

- a single accept state,
- no transitions to the start state,
- no transitions from the accept state, and
- each transition is labeled with a regular expression



GNFA acceptance

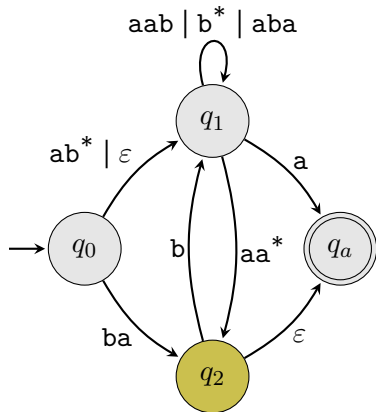
A GNFA transitions from one state to the next by reading a block of input symbols generated by the regex



babaaba

GNFA acceptance

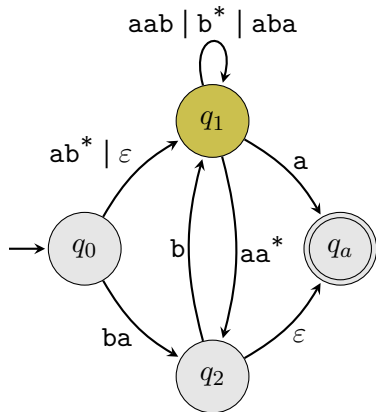
A GNFA transitions from one state to the next by reading a block of input symbols generated by the regex



ba**b**aaba

GNFA acceptance

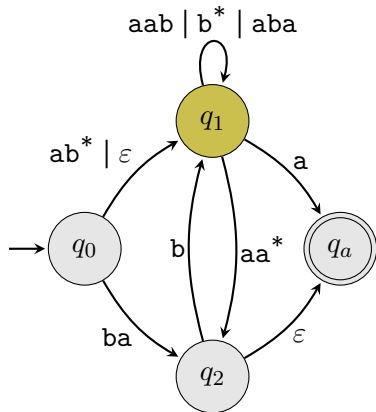
A GNFA transitions from one state to the next by reading a block of input symbols generated by the regex



bab**a**ba

GNFA acceptance

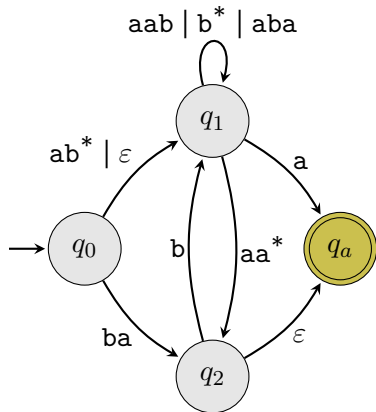
A GNFA transitions from one state to the next by reading a block of input symbols generated by the regex



babaab**a**

GNFA acceptance

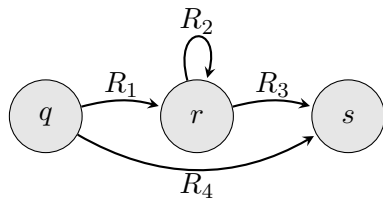
A GNFA transitions from one state to the next by reading a block of input symbols generated by the regex



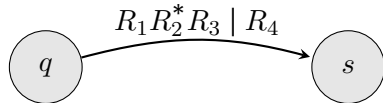
babaaba ✓ Accepted

Removing states in a GNFA

- 1 Select a state to remove r other than the start or accept states ($r \in Q \setminus \{q_0, q_a\}$)
- 2 For each $q, s \in Q \setminus \{r\}$ we have

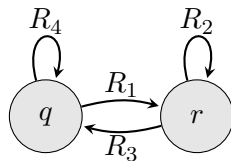
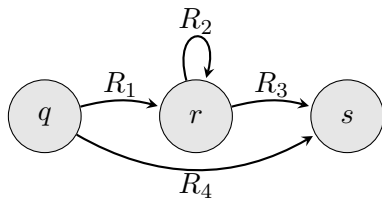


If a transition is missing from the GNFA, then the corresponding regex is \emptyset
Remove state r and replace regex R_4 with $R_1 R_2^* R_3 \mid R_4$

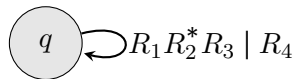
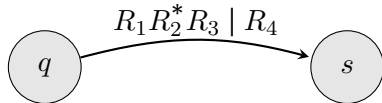


Removing states in a GNFA

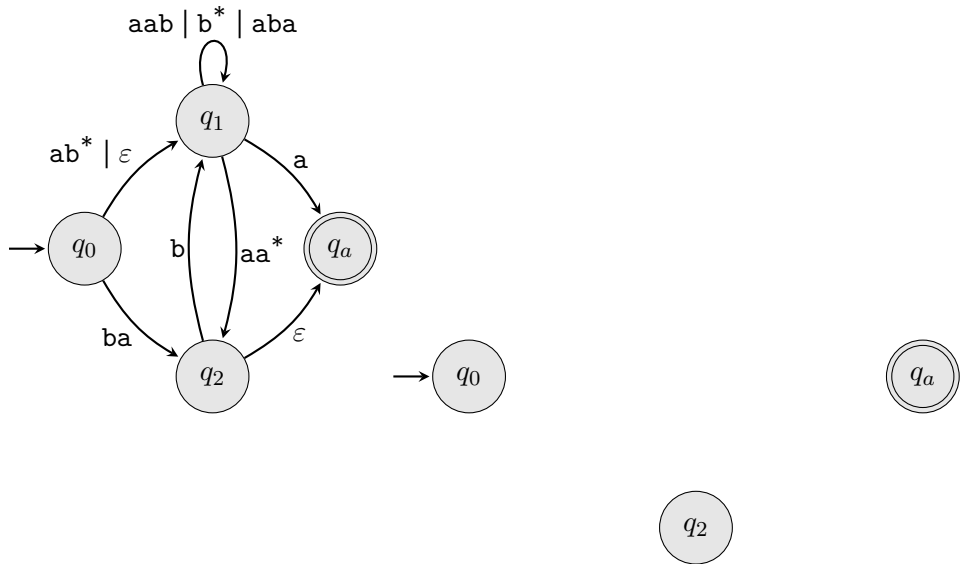
- 1 Select a state to remove r other than the start or accept states ($r \in Q \setminus \{q_0, q_a\}$)
- 2 For each $q, s \in Q \setminus \{r\}$ we have



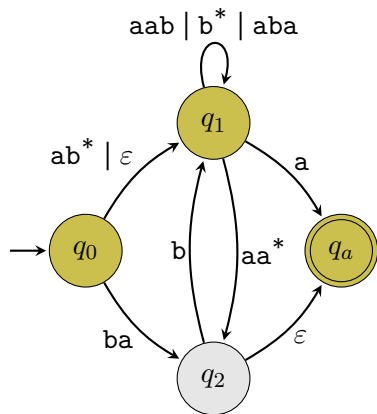
If a transition is missing from the GNFA, then the corresponding regex is \emptyset
 Remove state r and replace regex R_4 with $R_1 R_2^* R_3 \mid R_4$



Remove state q_1



Remove state q_1



$$R_1 = \underline{ab^* \mid \varepsilon}$$

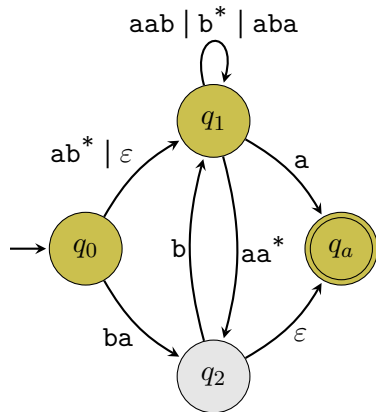
$$R_2 = \underline{aab \mid b^* \mid aba}$$

$$R_3 = \underline{a}$$

$$R_4 = \underline{\emptyset}$$



Remove state q_1

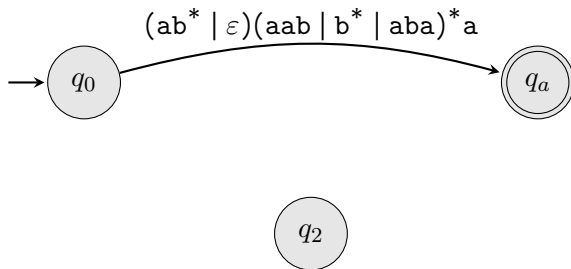


$$R_1 = \underline{ab^* \mid \varepsilon}$$

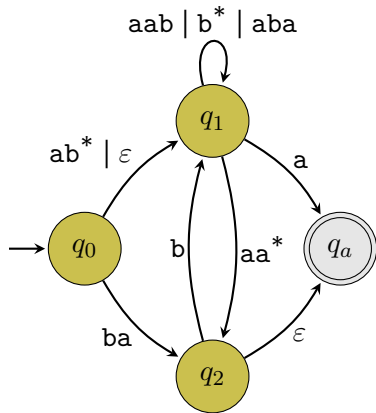
$$R_2 = \underline{aab \mid b^* \mid aba}$$

$$R_3 = \underline{a}$$

$$R_4 = \underline{\emptyset}$$



Remove state q_1

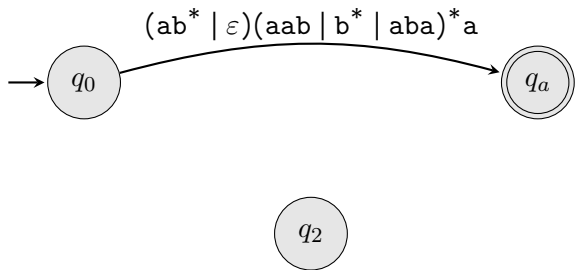


$$R_1 = \underline{ab^* \mid \varepsilon}$$

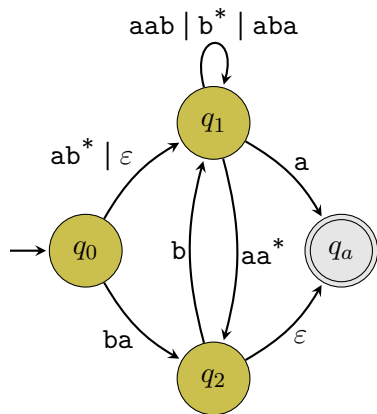
$$R_2 = \underline{aab \mid b^* \mid aba}$$

$$R_3 = \underline{aa^*}$$

$$R_4 = \underline{ba}$$



Remove state q_1

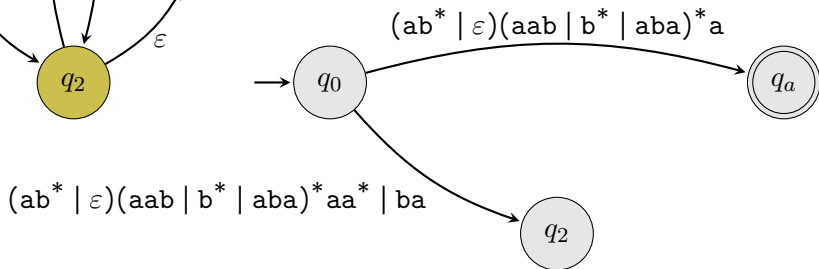


$$R_1 = \underline{ab^* \mid \varepsilon}$$

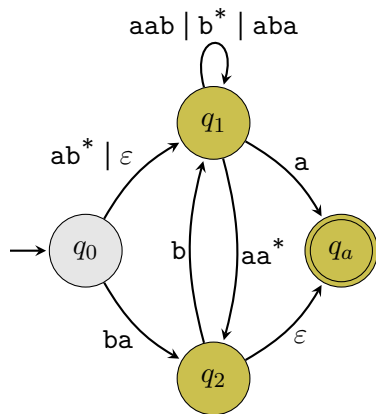
$$R_2 = \underline{aab \mid b^* \mid aba}$$

$$R_3 = \underline{aa^*}$$

$$R_4 = \underline{ba}$$



Remove state q_1

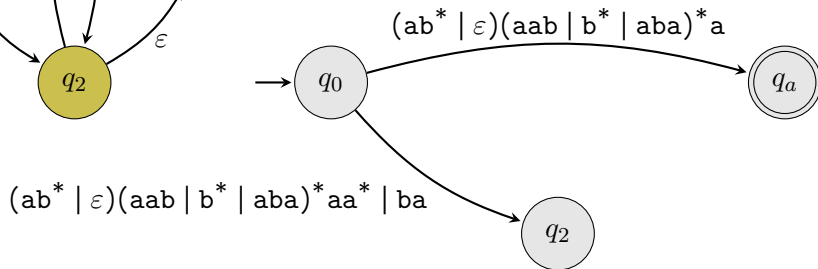


$$R_1 = \underline{b}$$

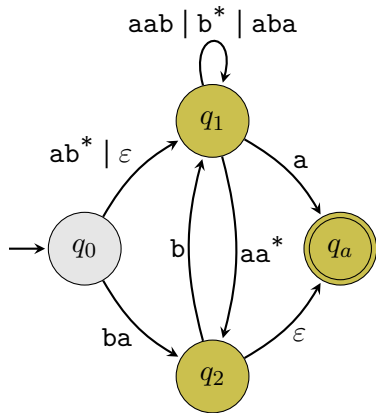
$$R_2 = \underline{aab \mid b^* \mid aba}$$

$$R_3 = \underline{a}$$

$$R_4 = \underline{\varepsilon}$$



Remove state q_1

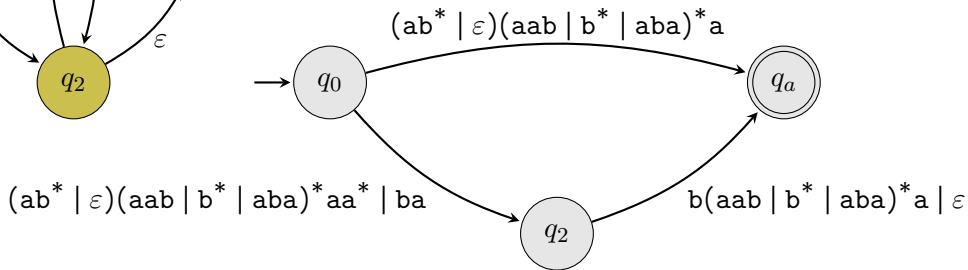


$$R_1 = \underline{b}$$

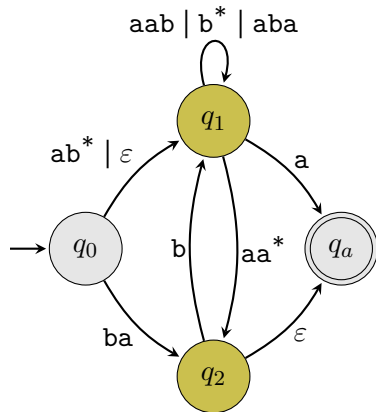
$$R_2 = \underline{aab \mid b^* \mid aba}$$

$$R_3 = \underline{a}$$

$$R_4 = \underline{\varepsilon}$$



Remove state q_1

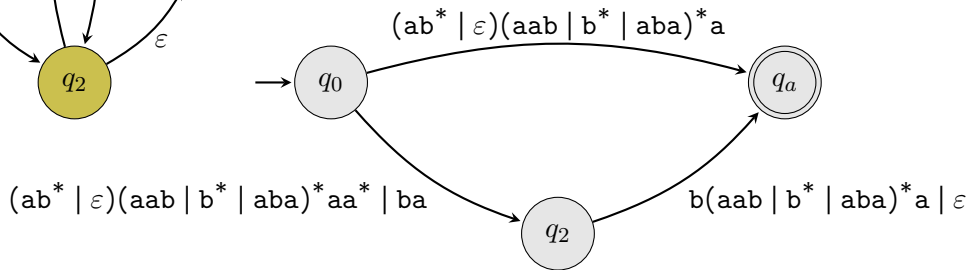


$$R_1 = \underline{b}$$

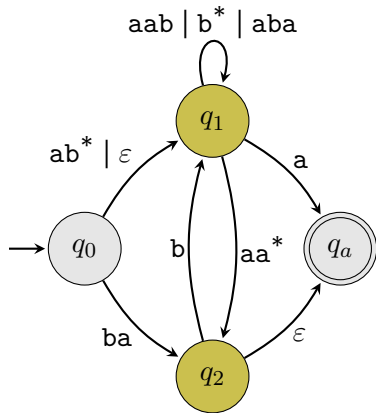
$$R_2 = \underline{aab \mid b^* \mid aba}$$

$$R_3 = \underline{aa^*}$$

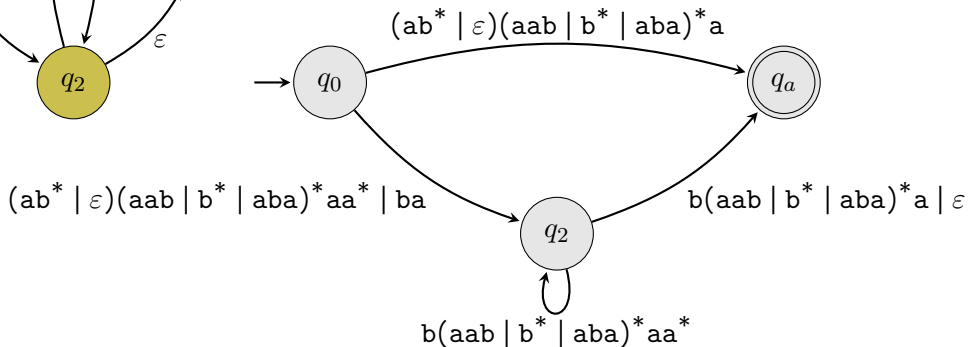
$$R_4 = \underline{\emptyset}$$



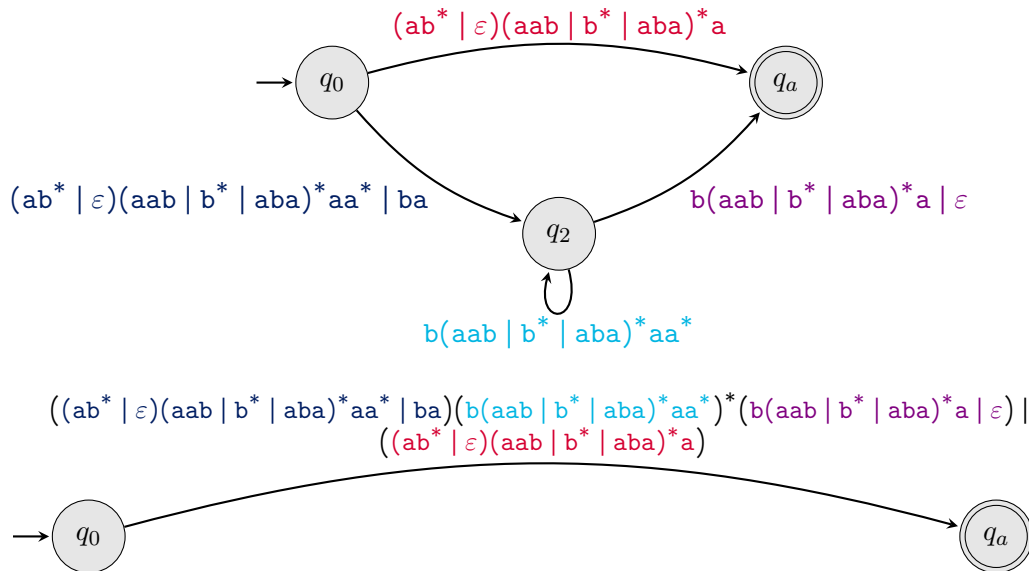
Remove state q_1



$$\begin{aligned}
 R_1 &= \underline{b} \\
 R_2 &= \underline{aab \mid b^* \mid aba} \\
 R_3 &= \underline{aa^*} \\
 R_4 &= \underline{\emptyset}
 \end{aligned}$$



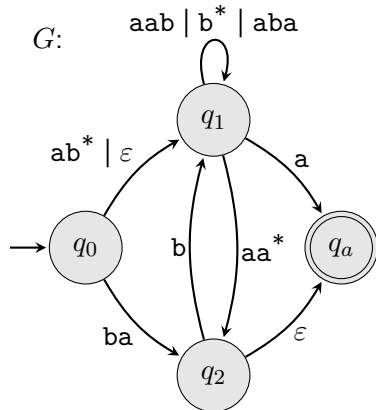
Remove state q_2



Converting GNFA to regular expression

Remove states one at a time until only the start and accept remain

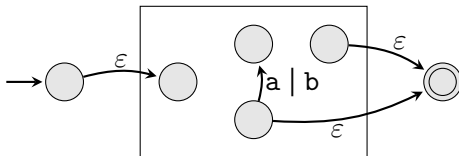
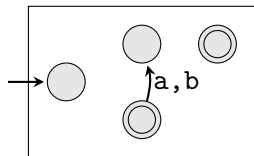
The one remaining transition is an equivalent regex



$$L(G) = \frac{((ab^* \mid \varepsilon)(aab \mid b^* \mid aba)^* aa^* \mid ba)(b(aab \mid b^* \mid aba)^* aa^*)^*(b(aab \mid b^* \mid aba)^* a \mid \varepsilon) \mid ((ab^* \mid \varepsilon)(aab \mid b^* \mid aba)^* a)}{1}$$

Converting an NFA (or DFA) to a GNFA

- 1 Add a new start state with an epsilon transition to the original start state
- 2 Add a new accept state with epsilon transitions from the original accept states
- 3 Convert multiple transitions between a pair of nodes to a single regex using $|$ to separate them



Converting an NFA (or DFA) to a regular expression

Theorem

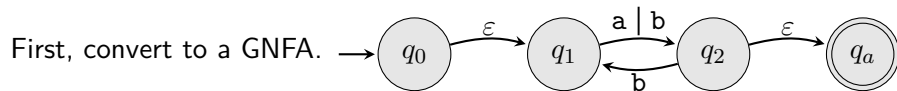
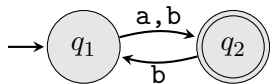
Every NFA (and thus every DFA) can be converted to an equivalent regular expression.

Proof. Given an NFA N , convert it to an equivalent GNFA G . Convert G to an equivalent regular expression.

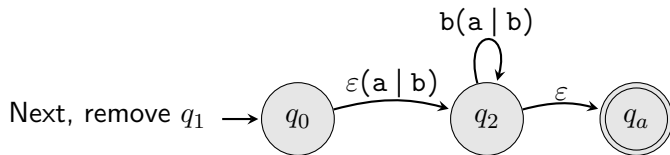
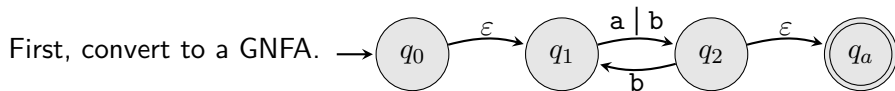
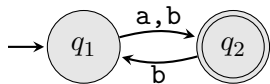
(Some details missing, but see the book.)



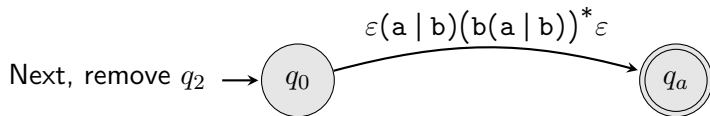
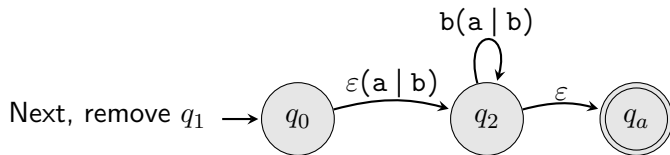
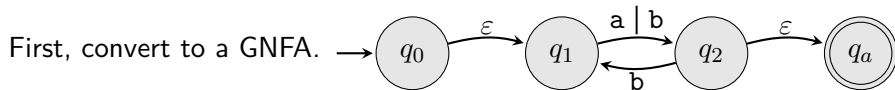
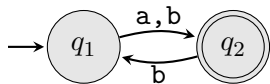
Example



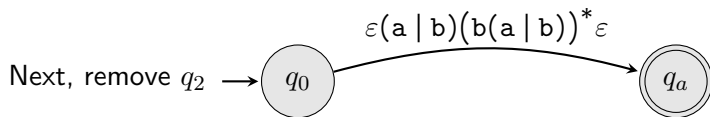
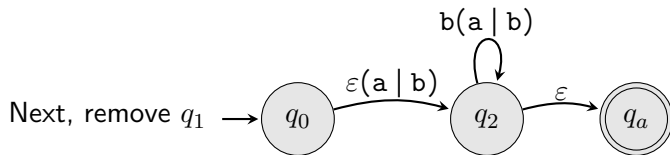
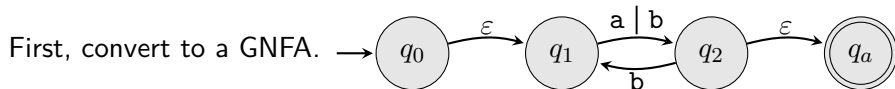
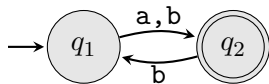
Example



Example

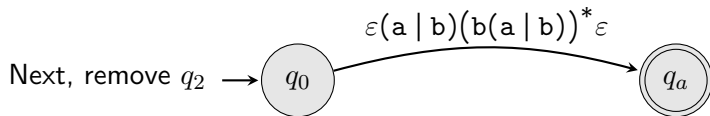
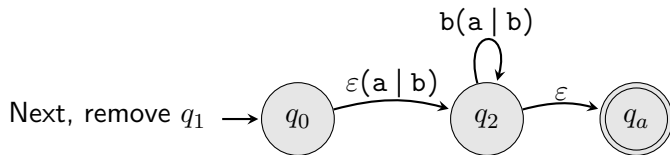
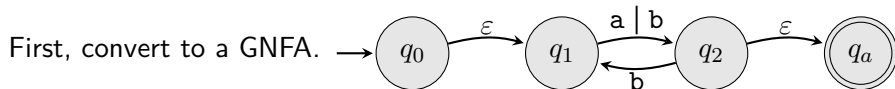
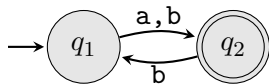


Example



Equivalent regular expression $\epsilon(a \mid b)(b(a \mid b))^* \epsilon$

Example



Equivalent regular expression $\epsilon(a \mid b)(b(a \mid b))^* \epsilon$ = $\Sigma(b\Sigma)^*$