CS 301

Lecture 04 – Regular Expressions

Stephen Checkoway

January 29, 2018

Review from last time

NFA $N = (Q, \Sigma, \delta, q_0, F)$ where $\delta : Q \times \Sigma_\varepsilon \to P(Q)$ 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

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

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

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

$$D = C^*$$

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

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

Describing complex languages using simpler ones

Use regular operations to break complex languages down into simpler ones

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

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

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

Describing complex languages using simpler ones

Use regular operations to break complex languages down into simpler ones

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

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

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

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

Describing complex languages using simpler ones

Use regular operations to break complex languages down into simpler ones

$$A = \{w \mid w \text{ starts and ends with the same symbols}\}$$
$$B = \{\mathtt{b}^k\mathtt{a} \mid k \geq 1\}$$
$$C = \{\varepsilon, \mathtt{ba}, \mathtt{aaa}\}$$

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

Describing complex languages using simpler ones

Use regular operations to break complex languages down into simpler ones

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

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

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

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

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

Describing complex languages using simpler ones

Use regular operations to break complex languages down into simpler ones

$$A = \{w \mid w \text{ starts and ends with the same symbols}\}$$
$$B = \{\mathtt{b}^k\mathtt{a} \mid k \geq 1\}$$
$$C = \{\varepsilon, \mathtt{ba}, \mathtt{aaa}\}$$

$$\begin{aligned} C &= \{\varepsilon\} \cup \{\mathtt{ba}\} \cup \{\mathtt{aaa}\} \\ &= \{\varepsilon\} \cup \big(\{\mathtt{b}\} \circ \{\mathtt{a}\}\big) \cup \big(\{\mathtt{a}\} \circ \{\mathtt{a}\} \circ \{\mathtt{a}\}\big) \\ B &= \{\mathtt{b}\} \circ \{\mathtt{b}\}^* \circ \{\mathtt{a}\} \\ A &= \{\mathtt{a}\} \cup \{\mathtt{b}\} \cup \big(\{\mathtt{a}\} \circ \Sigma^* \circ \{\mathtt{a}\}\big) \cup \big(\{\mathtt{b}\} \circ \Sigma^* \circ \{\mathtt{b}\}\big) \end{aligned}$$

Describing complex languages using simpler ones

Use regular operations to break complex languages down into simpler ones

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

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

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

$$\begin{aligned} C &= \{\varepsilon\} \cup \{\mathtt{ba}\} \cup \{\mathtt{aaa}\} \\ &= \{\varepsilon\} \cup \big(\{\mathtt{b}\} \circ \{\mathtt{a}\}\big) \cup \big(\{\mathtt{a}\} \circ \{\mathtt{a}\} \circ \{\mathtt{a}\}\big) \\ B &= \{\mathtt{b}\} \circ \{\mathtt{b}\}^* \circ \{\mathtt{a}\} \\ A &= \{\mathtt{a}\} \cup \{\mathtt{b}\} \cup \big(\{\mathtt{a}\} \circ \Sigma^* \circ \{\mathtt{a}\}\big) \cup \big(\{\mathtt{b}\} \circ \Sigma^* \circ \{\mathtt{b}\}\big) \\ &= \{\mathtt{a}\} \cup \{\mathtt{b}\} \cup \big(\{\mathtt{a}\} \circ (\{\mathtt{a}\} \cup \{\mathtt{b}\})^* \circ \{\mathtt{a}\}\big) \cup \big(\{\mathtt{b}\} \circ (\{\mathtt{a}\} \cup \{\mathtt{b}\})^* \circ \{\mathtt{b}\}\big) \end{aligned}$$

Describing complex languages using simpler ones

Use regular operations to break complex languages down into simpler ones

$$A = \{w \mid w \text{ starts and ends with the same symbols}\}$$
$$B = \{\mathtt{b}^k\mathtt{a} \mid k \geq 1\}$$
$$C = \{\varepsilon, \mathtt{ba}, \mathtt{aaa}\}$$

$$\begin{aligned}
C &= \{\varepsilon\} \cup \{\mathtt{ba}\} \cup \{\mathtt{aaa}\} \\
&= \{\varepsilon\} \cup \big(\{\mathtt{b}\} \circ \{\mathtt{a}\}\big) \cup \big(\{\mathtt{a}\} \circ \{\mathtt{a}\} \circ \{\mathtt{a}\}\big) \\
B &= \{\mathtt{b}\} \circ \{\mathtt{b}\}^* \circ \{\mathtt{a}\} \\
A &= \{\mathtt{a}\} \cup \{\mathtt{b}\} \cup \big(\{\mathtt{a}\} \circ \Sigma^* \circ \{\mathtt{a}\}\big) \cup \big(\{\mathtt{b}\} \circ \Sigma^* \circ \{\mathtt{b}\}\big) \\
&= \{\mathtt{a}\} \cup \{\mathtt{b}\} \cup \big(\{\mathtt{a}\} \circ (\{\mathtt{a}\} \cup \{\mathtt{b}\})^* \circ \{\mathtt{a}\}\big) \cup \big(\{\mathtt{b}\} \circ (\{\mathtt{a}\} \cup \{\mathtt{b}\})^* \circ \{\mathtt{b}\}\big)
\end{aligned}$$

We broke each language down into languages containing $\{\mathtt{a}\}$, $\{\mathtt{b}\}$, 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)

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

$$=$$

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

$$=$$

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

$$=$$

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)

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

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)

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

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)

$$
\begin{aligned}
A &= \{\mathtt{a}\} \cup \{\mathtt{b}\} \cup \big(\{\mathtt{a}\} \circ (\{\mathtt{a}\} \cup \{\mathtt{b}\})^* \circ \{\mathtt{a}\}\big) \cup \big(\{\mathtt{b}\} \circ (\{\mathtt{a}\} \cup \{\mathtt{b}\})^* \circ \{\mathtt{b}\}\big) \\
&= \underline{\mathtt{a} \mid \mathtt{b} \mid \mathtt{a}(\mathtt{a} \mid \mathtt{b})^*\mathtt{a} \mid \mathtt{b}(\mathtt{a} \mid \mathtt{b})^*\mathtt{b}} \\
B &= \{\mathtt{b}\} \circ \{\mathtt{b}\}^* \circ \{\mathtt{a}\} \\
&= \underline{\mathtt{b}\mathtt{b}^*\mathtt{a}} \\
C &= \{\varepsilon\} \cup \big(\{\mathtt{b}\} \circ \{\mathtt{a}\}\big) \cup \big(\{\mathtt{a}\} \circ \{\mathtt{a}\} \circ \{\mathtt{a}\}\big) \\
&= \underline{\varepsilon \mid \mathtt{ba} \mid \mathtt{aaa}}
\end{aligned}
$$

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)

$$
\begin{aligned}
A &= \{\mathtt{a}\} \cup \{\mathtt{b}\} \cup \big(\{\mathtt{a}\} \circ (\{\mathtt{a}\} \cup \{\mathtt{b}\})^* \circ \{\mathtt{a}\}\big) \cup \big(\{\mathtt{b}\} \circ (\{\mathtt{a}\} \cup \{\mathtt{b}\})^* \circ \{\mathtt{b}\}\big) \\
&= \underline{\mathtt{a} \mid \mathtt{b} \mid \mathtt{a}(\mathtt{a} \mid \mathtt{b})^*\mathtt{a} \mid \mathtt{b}(\mathtt{a} \mid \mathtt{b})^*\mathtt{b}} \\
B &= \{\mathtt{b}\} \circ \{\mathtt{b}\}^* \circ \{\mathtt{a}\} \\
&= \underline{\mathtt{bb}^*\mathtt{a}} \\
C &= \{\varepsilon\} \cup \big(\{\mathtt{b}\} \circ \{\mathtt{a}\}\big) \cup \big(\{\mathtt{a}\} \circ \{\mathtt{a}\} \circ \{\mathtt{a}\}\big) \\
&= \underline{\varepsilon \mid \mathtt{ba} \mid \mathtt{aaa}}
\end{aligned}
$$

Order of operation: *, $\circ$, $|$
Parentheses used for grouping
We underline the expression to differentiate the string $\mathtt{aaa}$ from the regular expression $\underline{\mathtt{aaa}}$

Regular expressions

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

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

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

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

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., $R = \underline{\texttt{ab}^*\texttt{a}}$ generates strings aa, aba, abba, ...
$L(R) = \{\texttt{ab}^k\texttt{a} \mid k \geq 0\}$

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 $\{\texttt{aba}\} = \underline{\texttt{aba}}$, that doesn't mean aba is the same as $\underline{\texttt{aba}}$!

Kleene star

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

Kleene star

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

Kleene star

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

Kleene star

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

Kleene star

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

Kleene star

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

Kleene star

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

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

Regular expression examples

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

Regular expression examples

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

Question 1

What strings are in the language given by the regular expression $\underline{(\texttt{a} \mid \texttt{bb})(\varepsilon \mid \texttt{a})}$?

Question 1

What strings are in the language given by the regular expression $\underline{(\texttt{a} \mid \texttt{bb})(\varepsilon \mid \texttt{a})}$?

`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: $A = B = \{\varepsilon, \mathtt{a}\}$. $A \circ B = \{\varepsilon, \mathtt{a}, \mathtt{aa}\}$

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

Question 3

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

Question 3

Is `abaaa` in the language given by $\underline{(\mathtt{a} \mid \mathtt{ba} \mid \mathtt{aaa})^*}$?

Yes. $\mathtt{abaaa} = \mathtt{a}\ \mathtt{ba}\ \mathtt{a}\ \mathtt{a}$

Question 4

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

Question 4

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

$\underline{\Sigma^*\texttt{baba}\Sigma^*}$

Question 5

Write a regex for the language
$\{w \mid \text{the second symbol of } w \text{ is } \mathtt{a} \text{ or the third to last symbol of } w \text{ is } \mathtt{b}\}$

Question 5

Write a regex for the language
$\{w \mid \text{the second symbol of } w \text{ is } \mathtt{a} \text{ or the third to last symbol of } w \text{ is } \mathtt{b}\}$

$\underline{\Sigma\mathtt{a}\Sigma^* \mid \Sigma^*\mathtt{b}\Sigma\Sigma}$

Question 6

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

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

Question 6

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

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

U.S. phone numbers.

We can rewrite this regex as

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

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

- $\underline{R_1 \mid \varnothing} = R_1$

Regular expression manipulation

Let R_1, R_2, and R_3 be regular expressions

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

Regular expression manipulation

Let R_1, R_2, and R_3 be regular expressions

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

Regular expression manipulation

Let R_1, R_2, and R_3 be regular expressions

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

Regular expression manipulation

Let R_1, R_2, and R_3 be regular expressions

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

Regular expression manipulation

Let R_1, R_2, and R_3 be regular expressions

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

Regular expression manipulation

Let R_1, R_2, and R_3 be regular expressions

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

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, $\underline{\varnothing}$, $\underline{\varepsilon}$ and $\underline{t}$ for $t \in \Sigma$

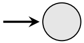
Then, we handle the three inductive cases, $\underline{R_1 \mid R_2}$, $\underline{R_1 \circ R_2}$, and $\underline{R_1^*}$

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.

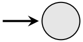
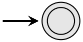
Base cases.

1. $R = \underline{\varnothing}$ 

Converting regular expressions to NFAs

Proof.

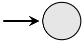
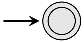
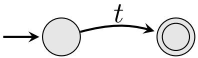
Base cases.

1. $R = \underline{\varnothing}$ 
2. $R = \underline{\varepsilon}$ 

Converting regular expressions to NFAs

Proof.

Base cases.

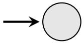
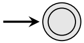
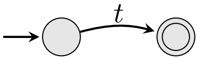
1. $R = \underline{\varnothing}$ 
2. $R = \underline{\varepsilon}$ 
3. $R = \underline{t}$ 

for $t \in \Sigma$

Converting regular expressions to NFAs

Proof.

Base cases.

1. $R = \underline{\varnothing}$ 
2. $R = \underline{\varepsilon}$ 
3. $R = \underline{t}$ 

for $t \in \Sigma$

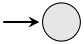
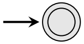
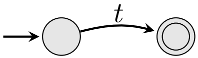
Inductive cases.

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.

1. $R = \underline{\varnothing}$ 
2. $R = \underline{\varepsilon}$ 
3. $R = \underline{t}$ 

for $t \in \Sigma$

Inductive cases.

4. $R = \underline{R_1 \mid R_2}$
5. $R = \underline{R_1 \circ R_2}$
6. $R = \underline{R_1^*}$

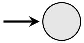
By the inductive hypothesis, there exist NFAs N_1 and N_2 such that $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.

Converting regular expressions to NFAs

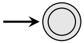
Proof.

Base cases.

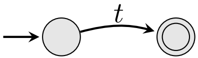
1. $R = \underline{\varnothing}$



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



3. $R = \underline{t}$



for $t \in \Sigma$

Inductive cases.

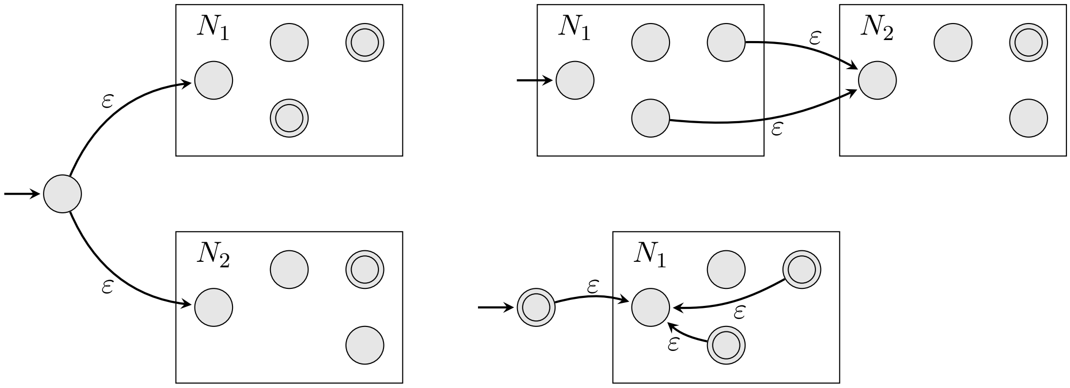
4. $R = \underline{R_1 \mid R_2}$
5. $R = \underline{R_1 \circ R_2}$
6. $R = \underline{R_1^*}$

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 $L(N) = L(R)$. $\square$

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:



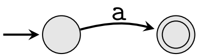
Regular expressions describe regular languages

The language of a regular expression is regular

This follow directly from the previous theorem:
Regular expression $\Rightarrow$ NFA $\Rightarrow$ DFA $\Rightarrow$ regular language

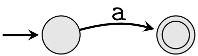
Regular expression to NFA: R = $\underline{\texttt{a(ba)}^* \mid \texttt{b(ab)}^*}$

1. $\underline{\texttt{a}}$

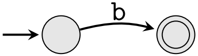


Regular expression to NFA: $R = \underline{\texttt{a(ba)}^* \mid \texttt{b(ab)}^*}$

1. $\underline{\texttt{a}}$

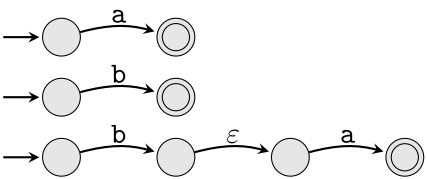


2. $\underline{\texttt{b}}$



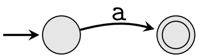
Regular expression to NFA: R = $\underline{\texttt{a(ba)}^* \mid \texttt{b(ab)}^*}$

1. $\underline{\texttt{a}}$
2. $\underline{\texttt{b}}$
3. $\underline{\texttt{ba}}$

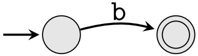


Regular expression to NFA: R = $\underline{\texttt{a(ba)}^* \mid \texttt{b(ab)}^*}$

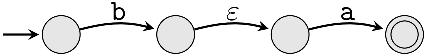
1. $\underline{\texttt{a}}$



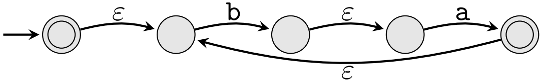
2. $\underline{\texttt{b}}$



3. $\underline{\texttt{ba}}$

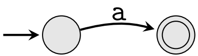


4. $\underline{\texttt{(ba)}^*}$

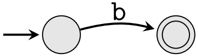


Regular expression to NFA: $R = \texttt{a(ba)}^* \mid \texttt{b(ab)}^*$

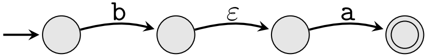
1. a



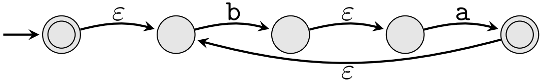
2. b



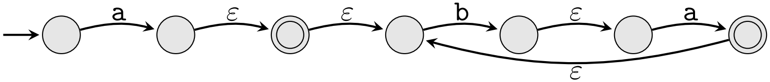
3. ba



4. $\texttt{(ba)}^*$

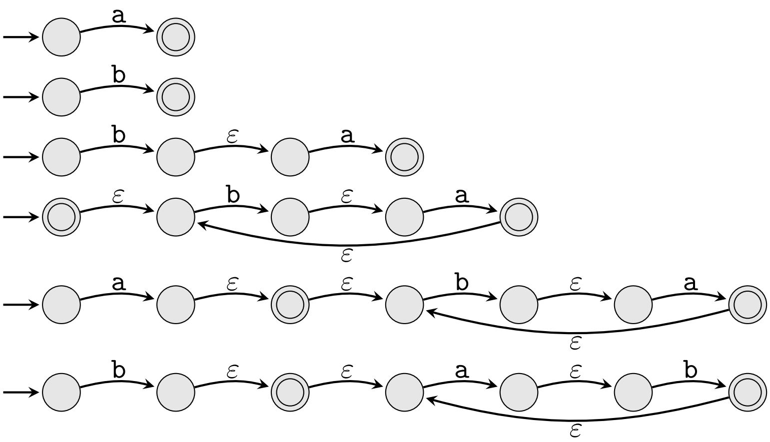


5. $\texttt{a(ba)}^*$



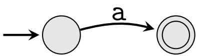
Regular expression to NFA: R = a(ba)* | b(ab)*

1. a
2. b
3. ba
4. (ba)*
5. a(ba)*
6. b(ab)*

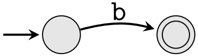


Regular expression to NFA: R = $\underline{\texttt{a(ba)}^* \mid \texttt{b(ab)}^*}$

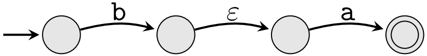
1. $\underline{\texttt{a}}$



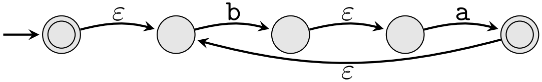
2. $\underline{\texttt{b}}$



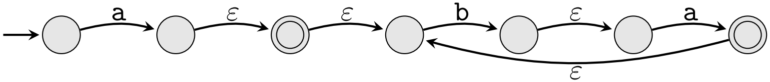
3. $\underline{\texttt{ba}}$



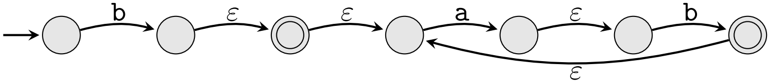
4. $\underline{\texttt{(ba)}^*}$



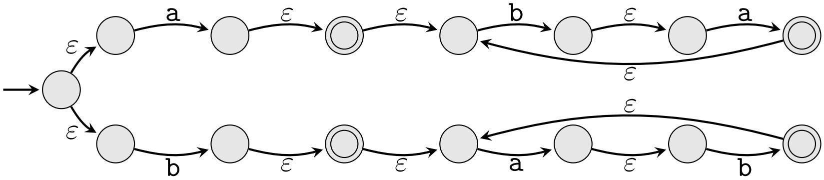
5. $\underline{\texttt{a(ba)}^*}$



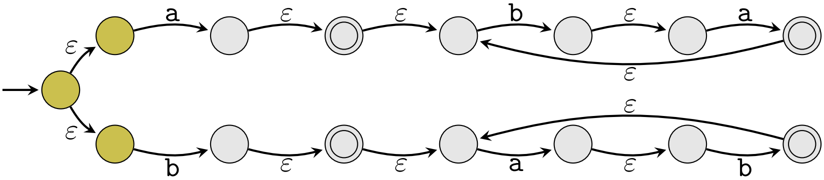
6. $\underline{\texttt{b(ab)}^*}$

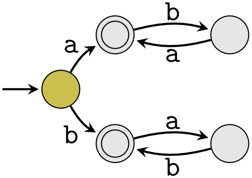


7. R



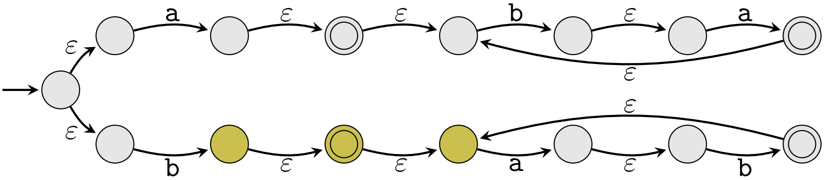
Not the smallest possible NFA

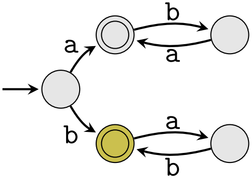




- babab

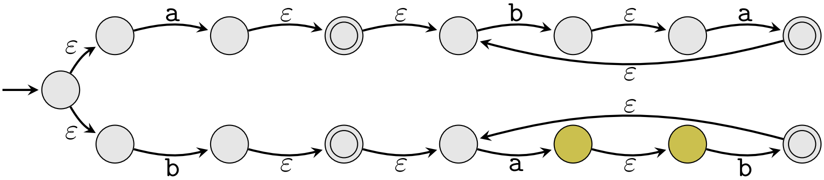
Not the smallest possible NFA

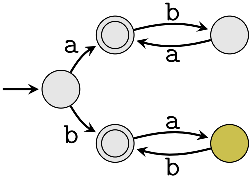




- babab

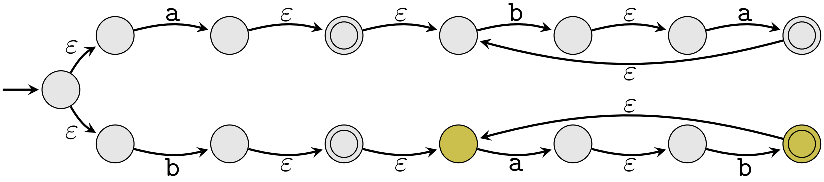
Not the smallest possible NFA

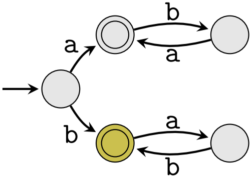




- babab

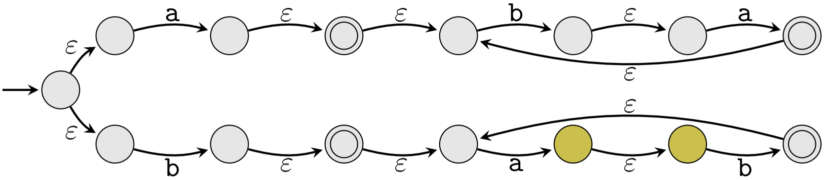
Not the smallest possible NFA

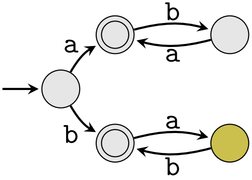




- babab

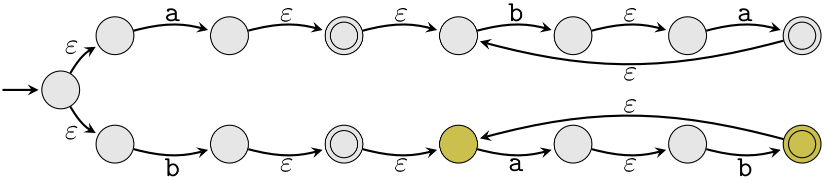
Not the smallest possible NFA

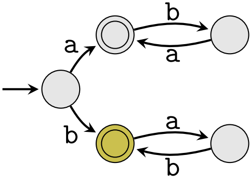




- babab

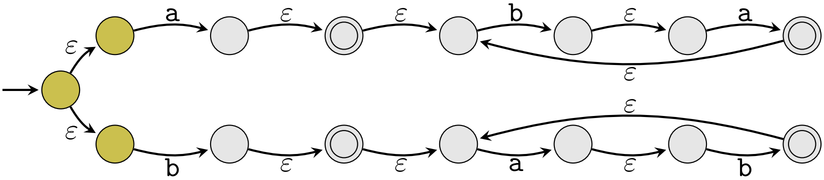
Not the smallest possible NFA

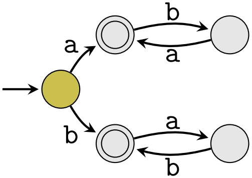




- babab ✔Accepted

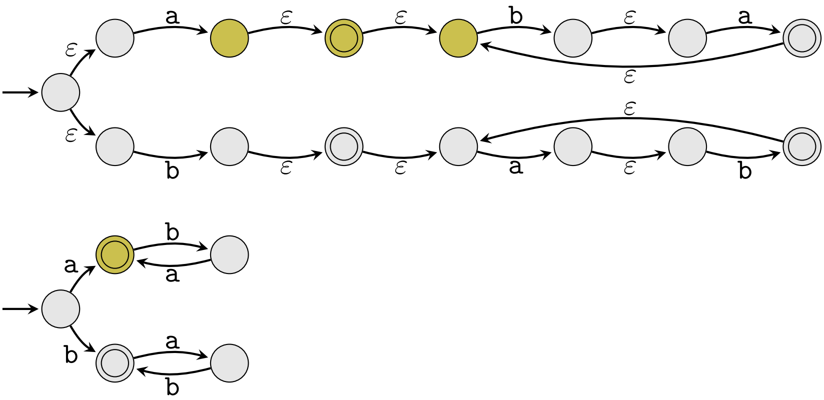
Not the smallest possible NFA





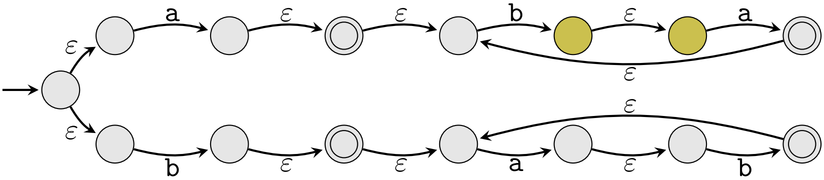
- babab ✔Accepted
- abab

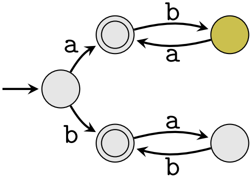
Not the smallest possible NFA



- `babab` ✔Accepted
- `abab`

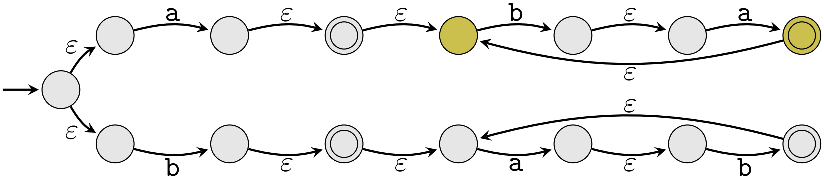
Not the smallest possible NFA

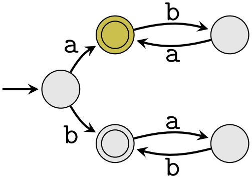




- `babab` ✔Accepted
- `abab`

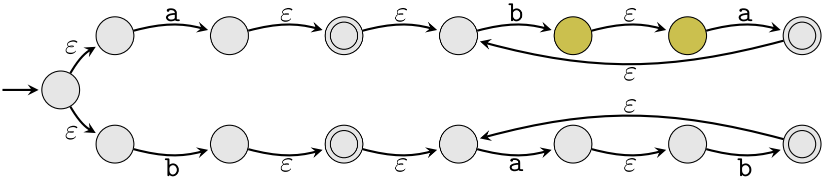
Not the smallest possible NFA

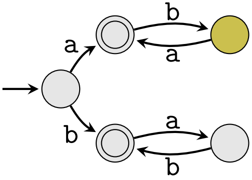




- babab ✔Accepted
- abab

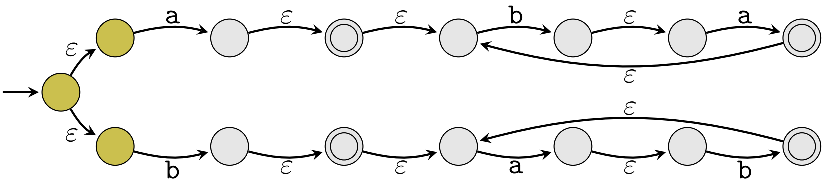
Not the smallest possible NFA

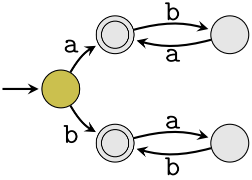




- babab ✔Accepted
- abab ✖Rejected

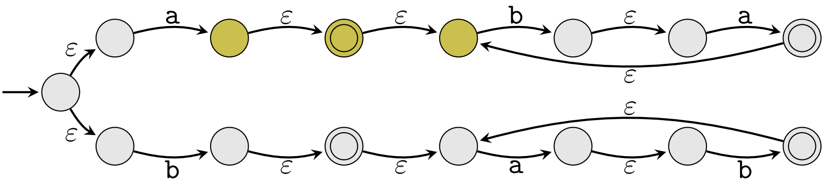
Not the smallest possible NFA

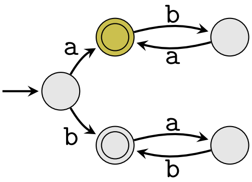




- babab ✔Accepted
- abab ✘Rejected
- abb

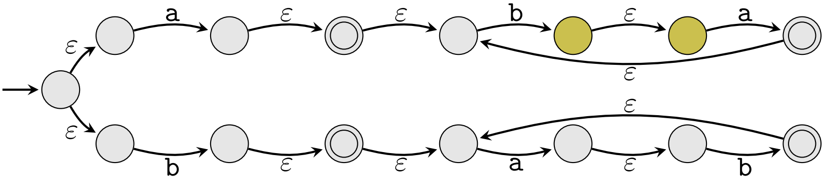
Not the smallest possible NFA

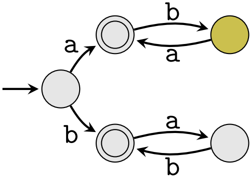




- babab ✔ Accepted
- abab ✖ Rejected
- abb

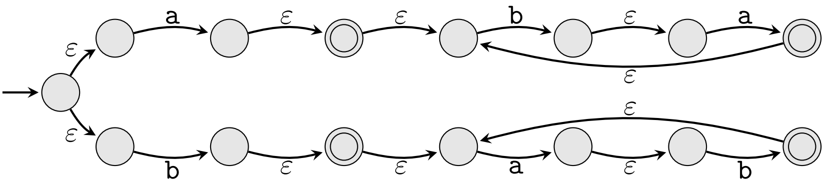
Not the smallest possible NFA

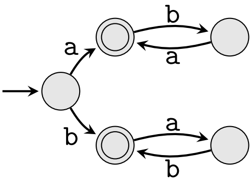




- babab ✔Accepted
- abab ✖Rejected
- abb

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.

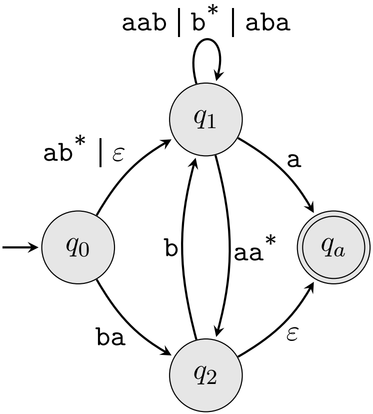
Proof idea

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)

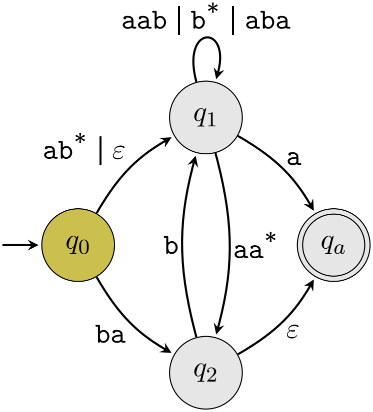
A GNFA is a finite automaton with

- 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

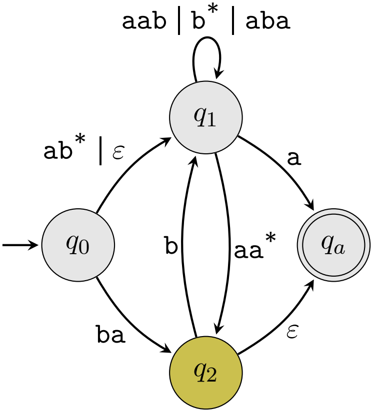
A GNFA transitions from one state to the next by reading a block of input symbols generated by the regex



babaaba

GNFA acceptance

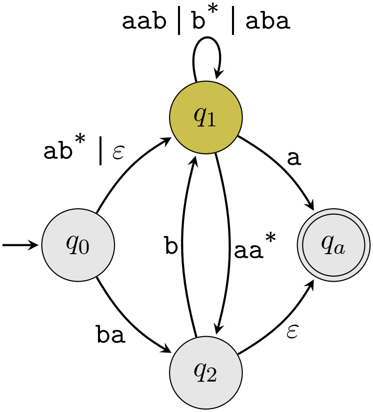
A GNFA transitions from one state to the next by reading a block of input symbols generated by the regex



babaaba

GNFA acceptance

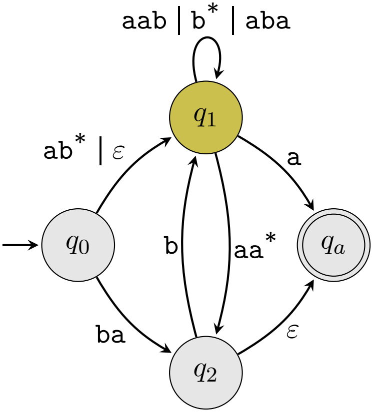
A GNFA transitions from one state to the next by reading a block of input symbols generated by the regex



babaaba

GNFA acceptance

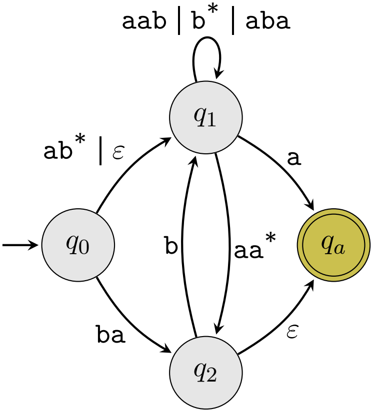
A GNFA transitions from one state to the next by reading a block of input symbols generated by the regex



babaaba

GNFA acceptance

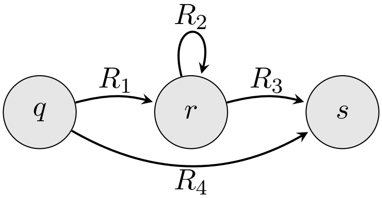
A GNFA transitions from one state to the next by reading a block of input symbols generated by the regex



babaaba ✔Accepted

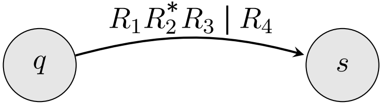
Removing states in a GNFA

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



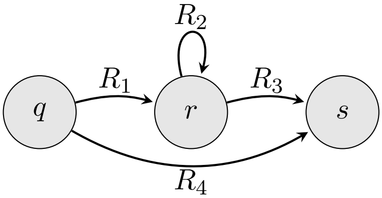
If a transition is missing from the GNFA, then the corresponding regex is $\underline{\varnothing}$

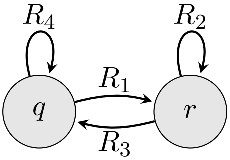
Remove state r and replace regex R_4 with $\underline{R_1 R_2{}^* R_3 \mid R_4}$



Removing states in a GNFA

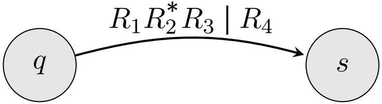
1. Select a state to remove r other than the start or accept states ($r \in Q \setminus \{q_0, q_a\}$)
2. For each $q, s \in Q \setminus \{r\}$ we have



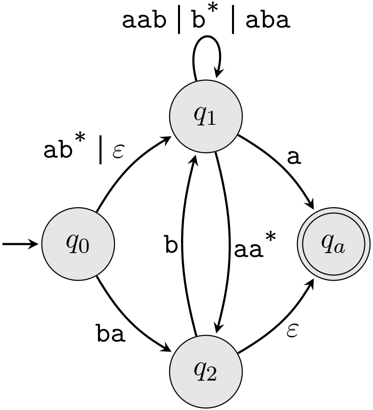


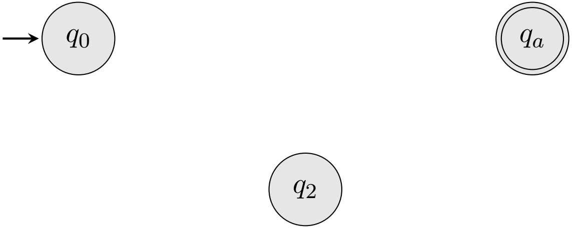
If a transition is missing from the GNFA, then the corresponding regex is $\underline{\emptyset}$

Remove state r and replace regex R_4 with $\underline{R_1 R_2{}^* R_3 \mid R_4}$

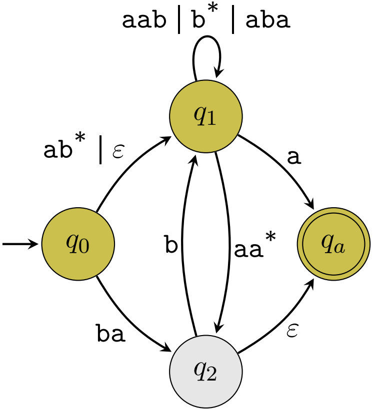


Remove state q_1





Remove state q_1

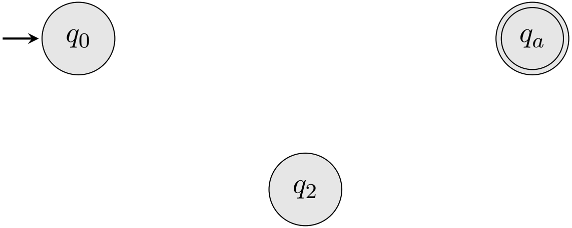


R_1 = $\underline{\texttt{ab}^* \mid \varepsilon}$

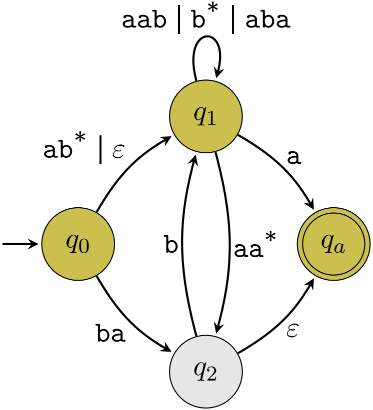
R_2 = $\underline{\texttt{aab} \mid \texttt{b}^* \mid \texttt{aba}}$

R_3 = $\underline{\texttt{a}}$

R_4 = $\underline{\varnothing}$



Remove state q_1

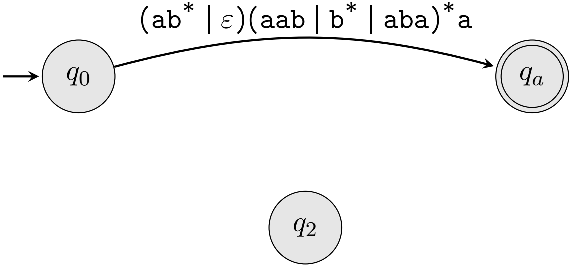


$$R_1 = \underline{\texttt{ab}^* \mid \varepsilon}$$

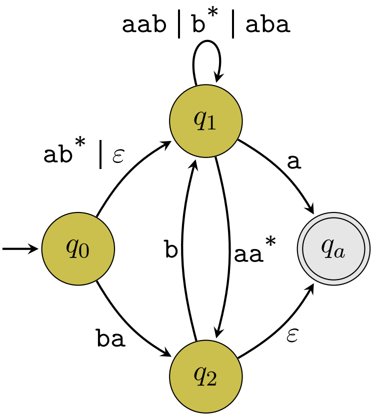
$$R_2 = \underline{\texttt{aab} \mid \texttt{b}^* \mid \texttt{aba}}$$

$$R_3 = \underline{\texttt{a}}$$

$$R_4 = \underline{\varnothing}$$



Remove state q_1

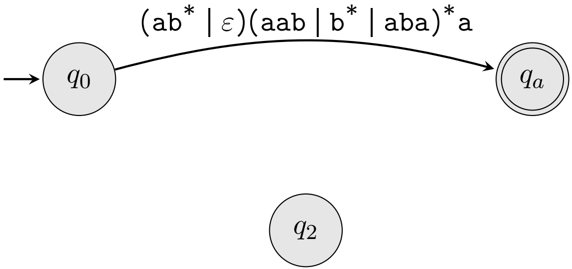


R_1 = ab* | ε

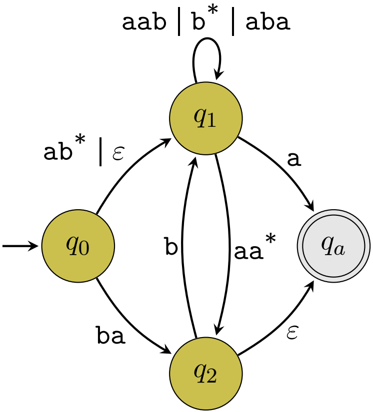
R_2 = aab | b* | aba

R_3 = aa*

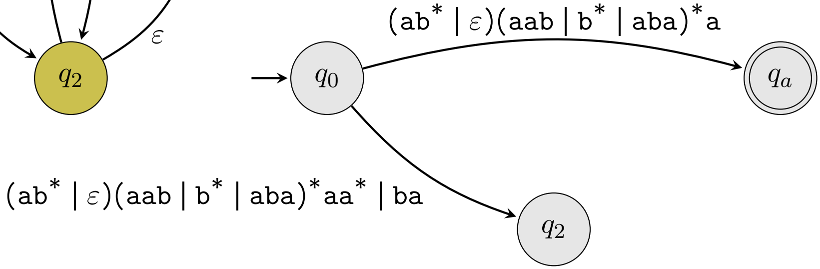
R_4 = ba



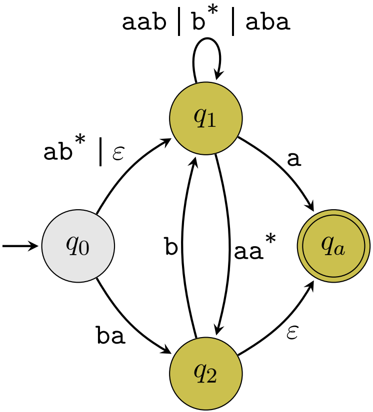
Remove state q_1



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



Remove state q_1

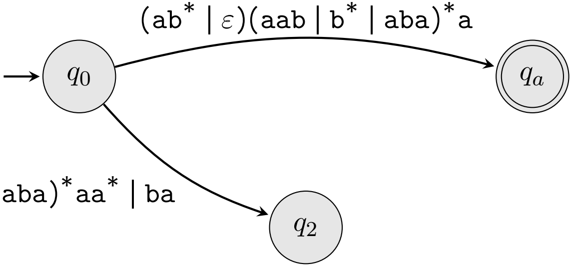


R_1 = $\underline{\texttt{b}}$

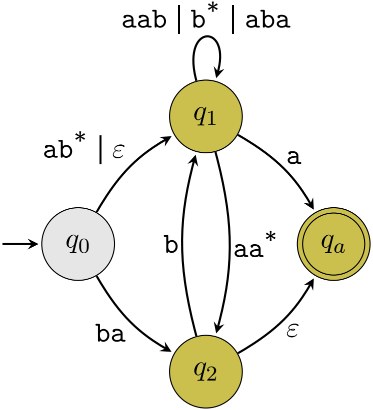
R_2 = $\underline{\texttt{aab} \mid \texttt{b}^* \mid \texttt{aba}}$

R_3 = $\underline{\texttt{a}}$

R_4 = $\underline{\varepsilon}$



Remove state q_1

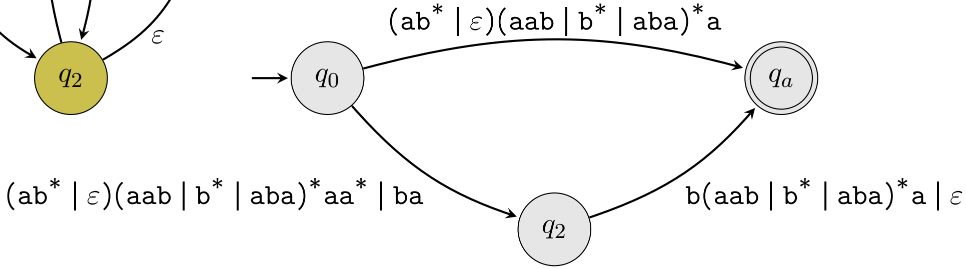


R_1 = b

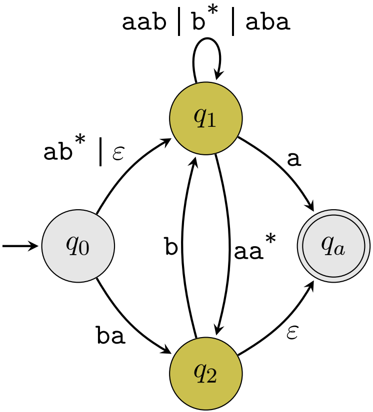
R_2 = aab | b* | aba

R_3 = a

R_4 = ε



Remove state q_1

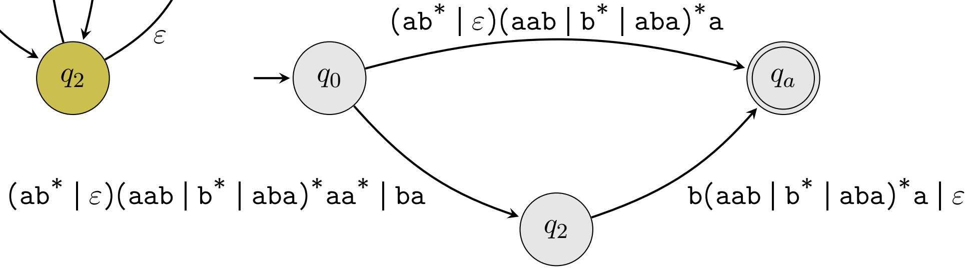


$R_1 = \underline{\texttt{b}}$

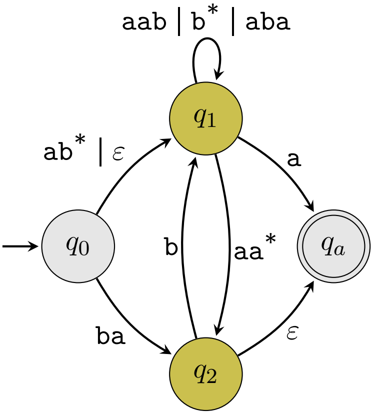
$R_2 = \underline{\texttt{aab} \mid \texttt{b}^* \mid \texttt{aba}}$

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

$R_4 = \underline{\varnothing}$



Remove state q_1

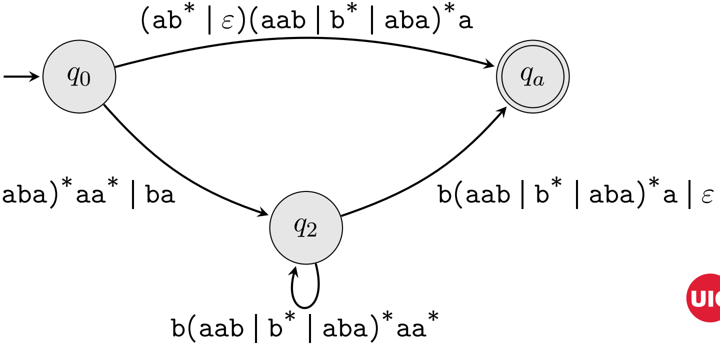


$$R_1 = \underline{\texttt{b}}$$

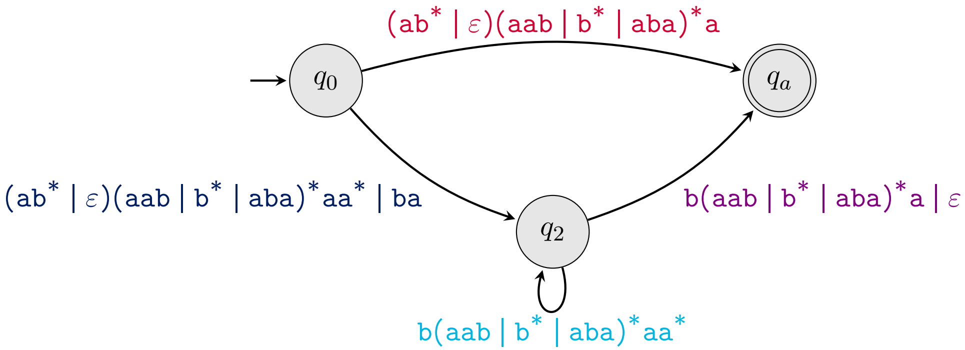
$$R_2 = \underline{\texttt{aab} \mid \texttt{b}^* \mid \texttt{aba}}$$

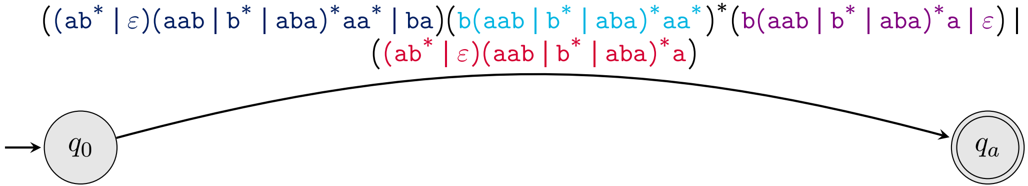
$$R_3 = \underline{\texttt{aa}^*}$$

$$R_4 = \underline{\varnothing}$$



Remove state q_2



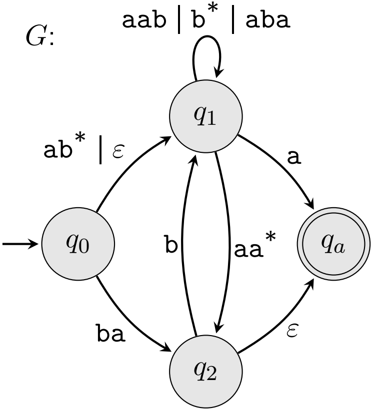


Converting GNFA to regular expression

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

The one remaining transition is an equivalent regex

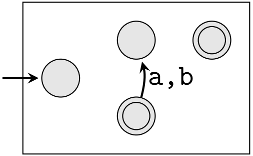
G:

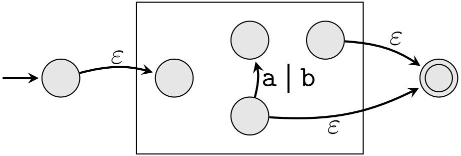


$$L(G) = \underline{\big((\mathtt{ab}^* \mid \varepsilon)(\mathtt{aab} \mid \mathtt{b}^* \mid \mathtt{aba})^*\mathtt{aa}^* \mid \mathtt{ba}\big)\big(\mathtt{b}(\mathtt{aab} \mid \mathtt{b}^* \mid \mathtt{aba})^*\mathtt{aa}^*\big)^*\big(\mathtt{b}(\mathtt{aab} \mid \mathtt{b}^* \mid \mathtt{aba})^*\mathtt{a}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

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

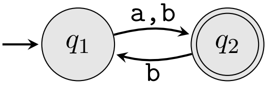
Every NFA (and thus every DFA) can be converted to an equivalent regular expression.

Proof.

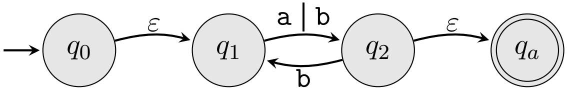
Given an NFA N, convert it to an equivalent GNFA G. Convert G to an equivalent regular expression.

(Some details missing, but see the book.) $\square$

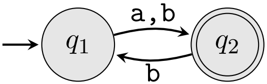
Example



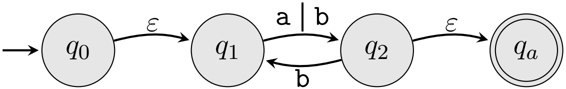
First, convert to a GNFA.



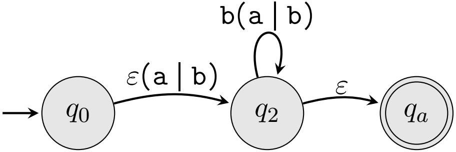
Example



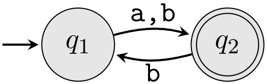
First, convert to a GNFA.



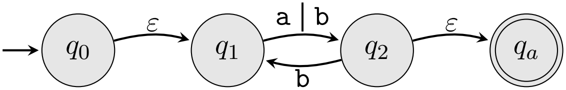
Next, remove q_1



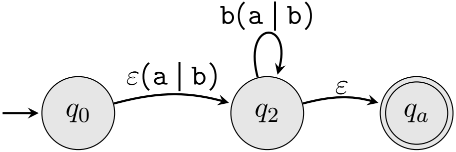
Example



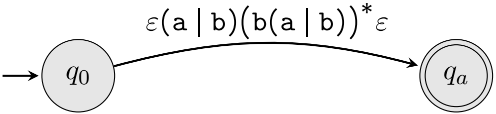
First, convert to a GNFA.



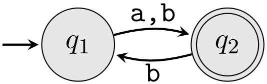
Next, remove q_1



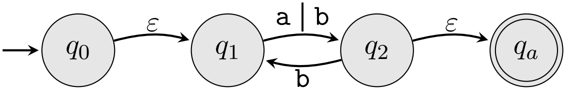
Next, remove q_2



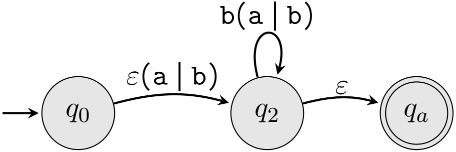
Example



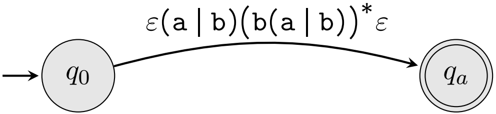
First, convert to a GNFA.



Next, remove q_1

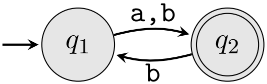


Next, remove q_2

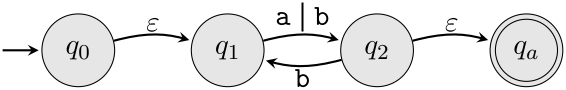


Equivalent regular expression $\underline{\varepsilon(\texttt{a}\mid\texttt{b})(\texttt{b}(\texttt{a}\mid\texttt{b}))^*\varepsilon}$

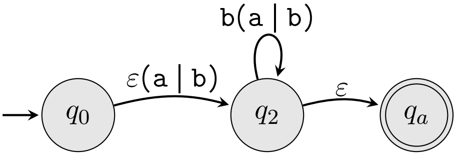
Example



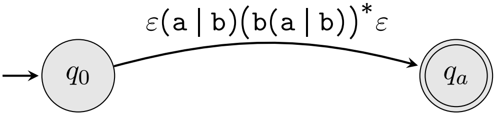
First, convert to a GNFA.



Next, remove q_1



Next, remove q_2



Equivalent regular expression $\underline{\varepsilon(\texttt{a} \mid \texttt{b})\big(\texttt{b}(\texttt{a} \mid \texttt{b})\big)^*\varepsilon} = \underline{\Sigma(\texttt{b}\Sigma)^*}$