

# BU CS 332 – Theory of Computation

<https://forms.gle/vYGrkBwBR22D8w8L8>



## Lecture 6:

- **Regexes = NFAs**
- **Limitations of Finite Automata**

Reading:

Sipser Ch 1.3

“Myhill-Nerode” note

Mark Bun & Alexander Poremba

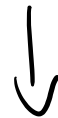
February 5, 2026

# Recap: Languages & Regular Expressions

$$\Sigma = \{0,1\} \quad L \subseteq \Sigma^*$$

$$\begin{aligned} \text{Ex: } L &= \{w \mid w \text{ contains exactly one } 1\} \\ &= \{x1y \mid x, y \in \{0\}^*\} \end{aligned}$$

Regular expression:  $0^*10^*$



$$\text{language } L(0^*10^*) = L$$

# Regular Expressions – Syntax

A regular expression  $R$  is defined recursively using the following rules:

1.  $\varepsilon$ ,  $\emptyset$ , and  $a$  are regular expressions for every  $a \in \Sigma$
2. If  $R_1$  and  $R_2$  are regular expressions, then so are  $(R_1 \cup R_2)$ ,  $(R_1 \circ R_2)$ , and  $(R_1^*)$

Examples: (over  $\Sigma = \{a, b, c\}$ ) (with simplified notation)

$ab$                        $ab^*c \cup (a^*b)^*$                        $\emptyset$

# Regular Expressions – Semantics

$L(R)$  = the language a regular expression describes

1.  $L(\emptyset) = \emptyset$
2.  $L(\varepsilon) = \{\varepsilon\}$
3.  $L(a) = \{a\}$  for every  $a \in \Sigma$
4.  $L((R_1 \cup R_2)) = L(R_1) \cup L(R_2)$
5.  $L((R_1 \circ R_2)) = L(R_1) \circ L(R_2)$
6.  $L((R_1^*)) = (L(R_1))^*$

**Example:**  $L(a^*b^*) = \{a^m b^n \mid m, n \geq 0\}$

# Syntactic Sugar

- For alphabet  $\Sigma$ , the regex  $\Sigma$  represents  $L(\Sigma) = \Sigma$

$$\Sigma = \{a, b, c\}$$

$$\Sigma = a \cup b \cup c$$

- For regex  $R$ , the regex  $R^+ = RR^*$

"One or more concatenations  
of strings from  $R$ "

# Regexes in the Real World

`grep` = globally search for a regular expression and print matching lines

```
$ grep '^xy*z' myfile
xyz
xyzde
xz
xz
xyz
xyyz
xyyyz
xyyyyz
$ grep '^x.*z' myfile
xyz
xyzde
xxz
xzz
x\z
x*z
xz
x z
xYz
xyyz
xyyyz
xyyyyz
$ grep '^x\*z' myfile
x*z
$ grep '\\\*' myfile
x\z
$
```

# Regular Expressions Describe Regular Languages

**Theorem:** A language  $A$  is regular if and only if it is described by a regular expression

← recognized by a DFA  
↔ recognized by an NFA

**Theorem 1:** Every regular expression has an equivalent NFA

**Theorem 2:** Every NFA has an equivalent regular expression

# Regular expression $\rightarrow$ NFA

**Theorem 1:** Every regex has an equivalent NFA

Proof: Induction on size of a regex

Base cases:	$L(R)$	NFA
$R = \emptyset$	$\emptyset$	$\rightarrow \bigcirc$
$R = \varepsilon$	$\{\varepsilon\}$	$\rightarrow \bigcirc$
$R = a$	$\{a\}$	$\rightarrow \bigcirc \xrightarrow{a} \bigcirc$

# Regular expression $\rightarrow$ NFA

**Theorem 1:** Every regex has an equivalent NFA

Proof: Induction on size of a regex

# of symbols :  $\phi, \epsilon, a, b, (, ), \cdot, \cup, *, \dots$



What should the inductive hypothesis be?

- a) Suppose **some** regular expression of length  $k$  can be converted to an NFA
- b) Suppose **every** regular expression of length  $k$  can be converted to an NFA
- c) Suppose **every** regular expression of length **at most**  $k$  can be converted to an NFA
- d) None of the above

# Regular expression $\rightarrow$ NFA

**Theorem 1:** Every regex has an equivalent NFA

Proof: Induction on size of a regex. Base case  $\checkmark$

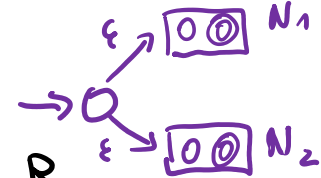
IH: "Assume every regex of size  $\leq k$  has an equivalent NFA."

WTS: If  $R$  is a regex of size  $k+1$ , then  $R$  has an equiv. NFA.

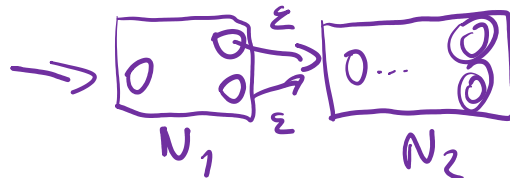
Inductive step:

$$R = (R_1 \cup R_2) \rightarrow L(R) = L(R_1) \cup L(R_2)$$

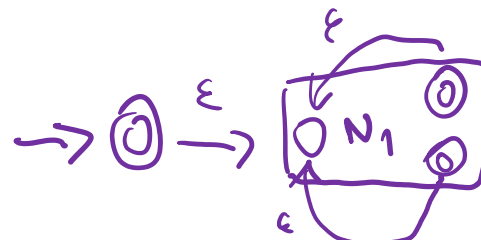
By IH:  $\Rightarrow \exists N_1$  recognizing  $R_1$   
 $\exists N_2$  recognizing  $R_2$



$$R = (R_1 R_2)$$



$$R = (R_1^*)$$





# Regular Expressions Describe Regular Languages

**Theorem:** A language  $A$  is regular if and only if it is described by a regular expression

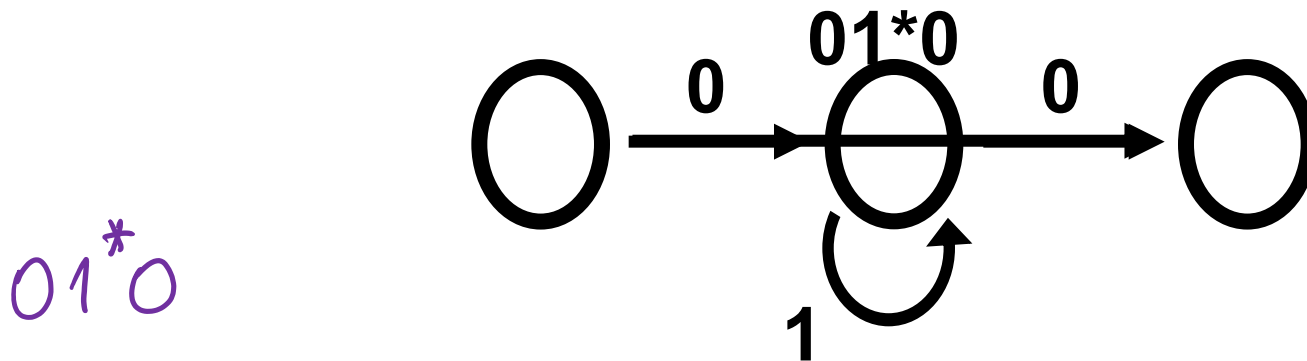
**Theorem 1:** Every regular expression has an equivalent NFA

**Theorem 2:** Every NFA has an equivalent regular expression

# NFA $\rightarrow$ Regular expression

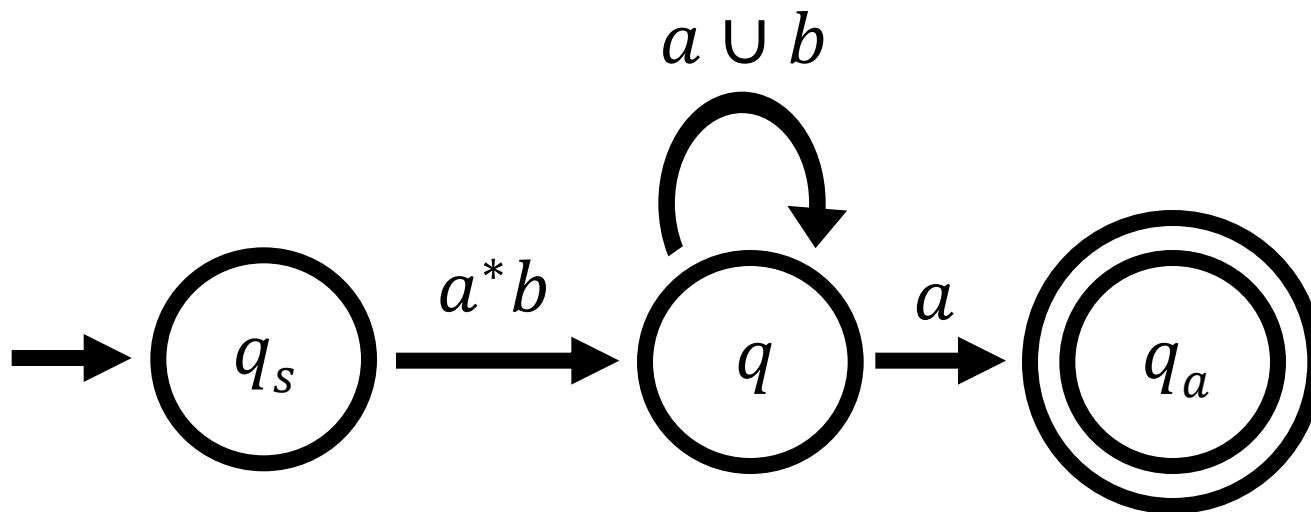
**Theorem 2:** Every NFA has an equivalent regex

Proof idea: Simplify NFA by “ripping out” states one at a time and replacing with regexes

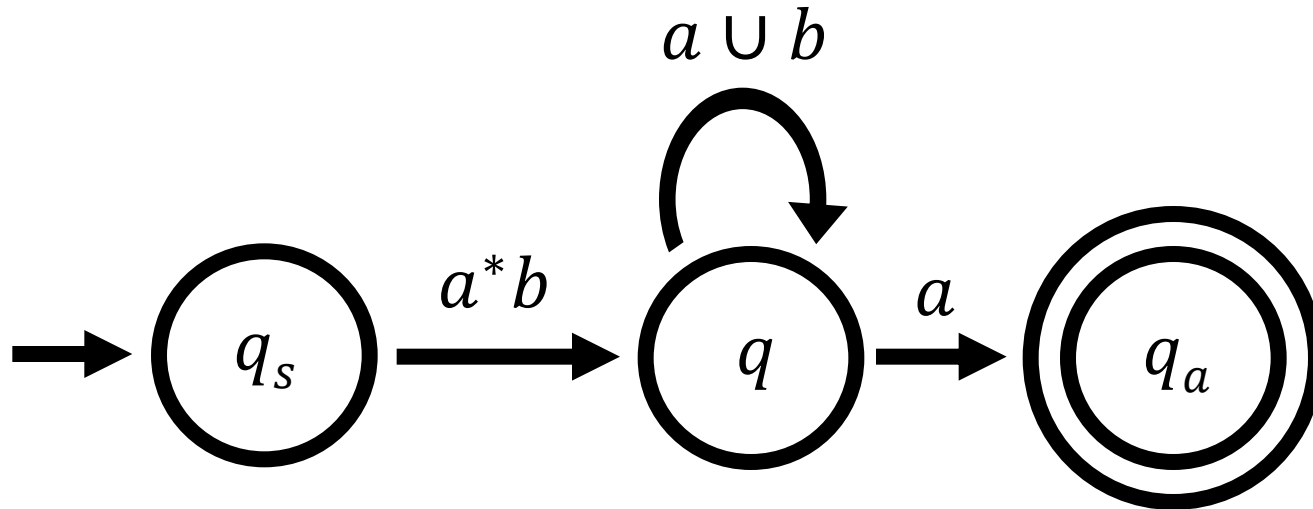


# Generalized NFAs

- **Every transition is labeled by a regex**
- One start state with only outgoing transitions
- Only one accept state with only incoming transitions
- Start state and accept state are distinct



# Generalized NFA Example



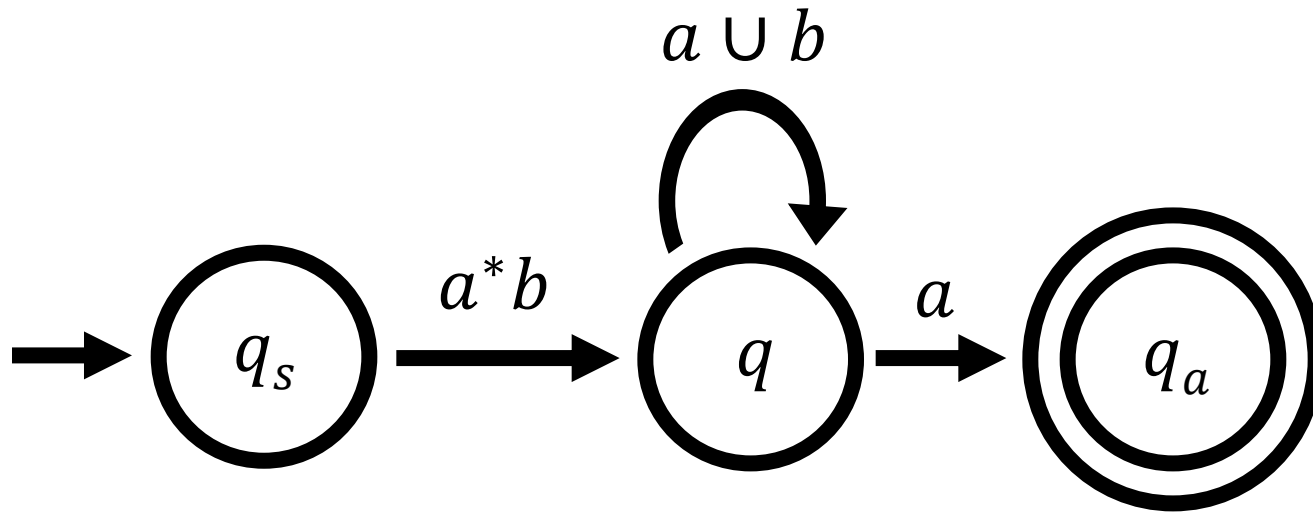
$$R(q_s, q) = a^*b$$

$$R(q_a, q) = \emptyset$$

$$R(q, q_s) = \emptyset$$

# Which of these strings is accepted?

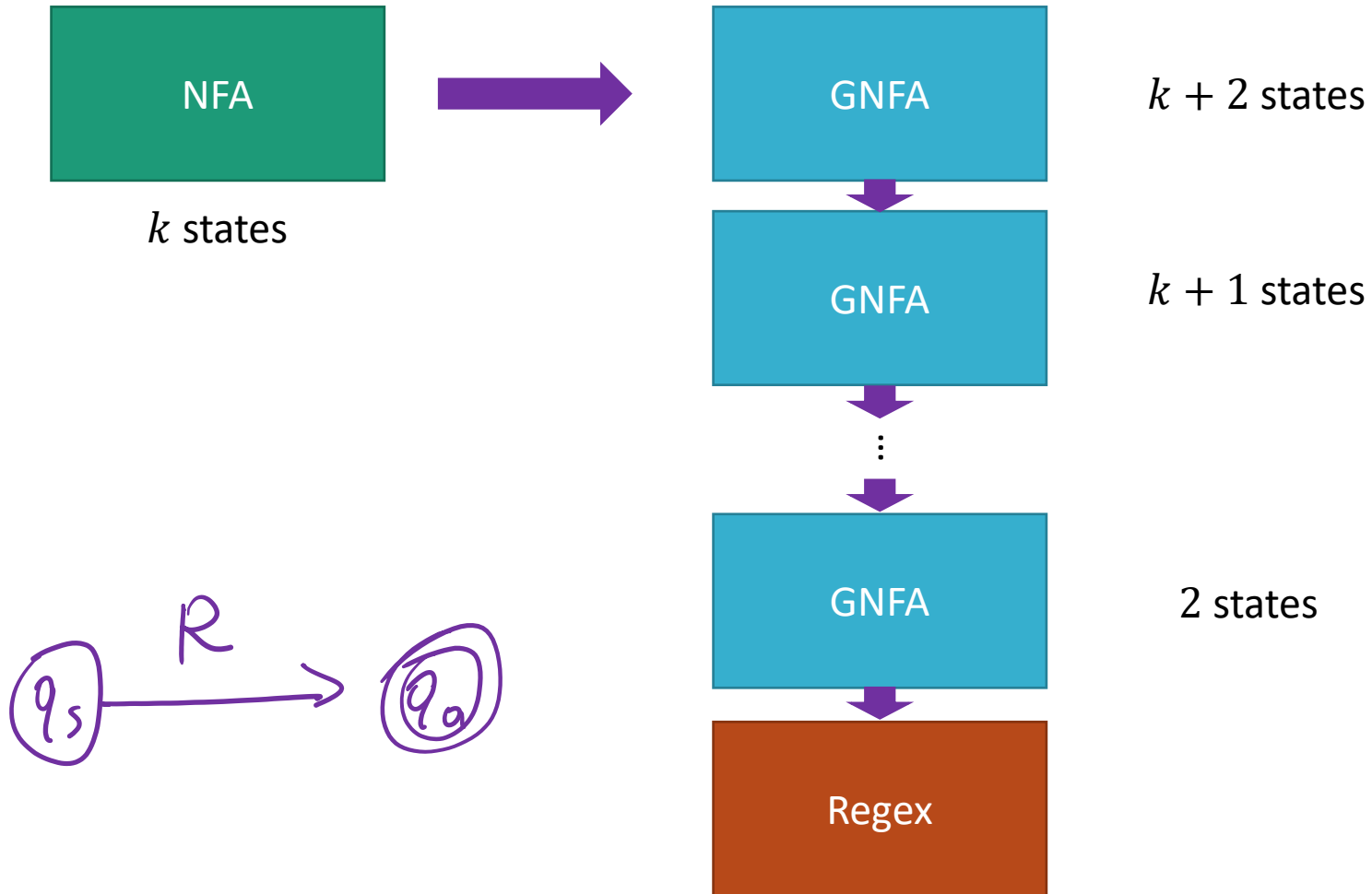
Which of the following strings is accepted by this GNFA?



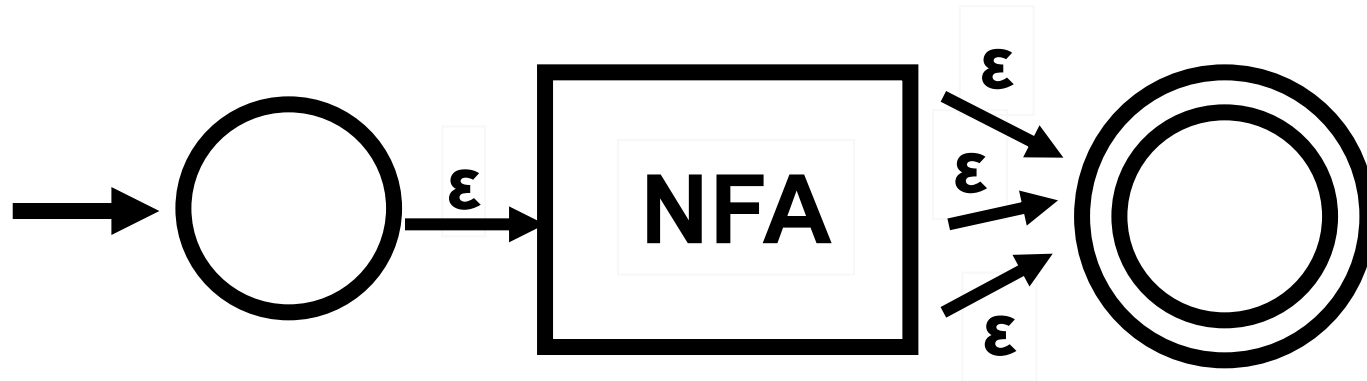
- a)  $aaa$
- b)  $aabb$
- c)  $bbb$
- d)  $bba$



# NFA $\rightarrow$ Regular expression



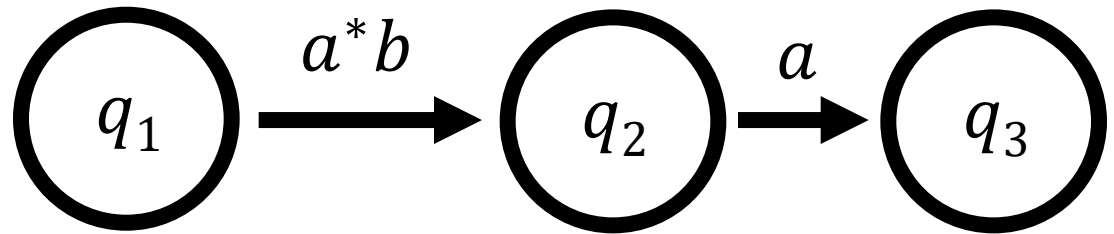
# NFA $\rightarrow$ GNFA



- Add a new start state with no incoming arrows.
- Make a unique accept state with no outgoing arrows.

# GNFA $\rightarrow$ Regular expression

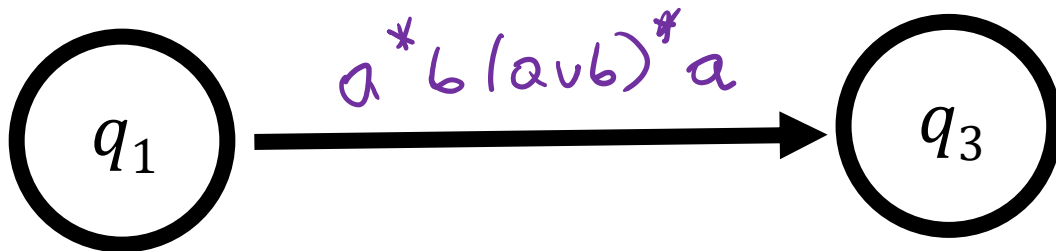
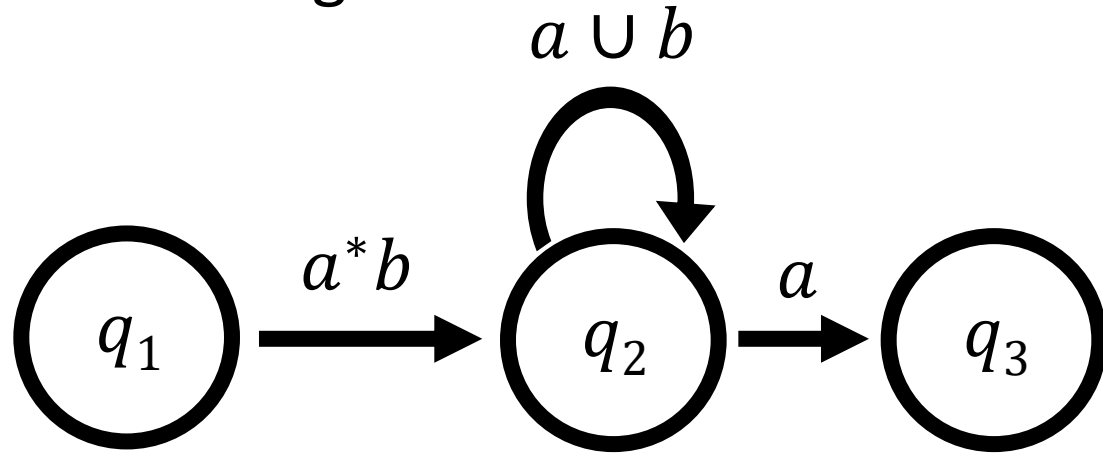
**Idea:** While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state



# GNFA $\rightarrow$ Regular expression

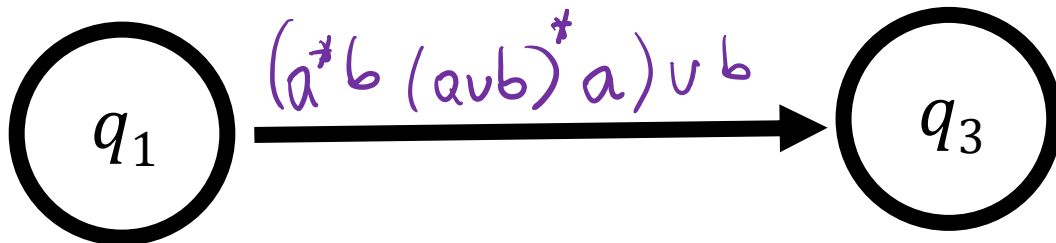
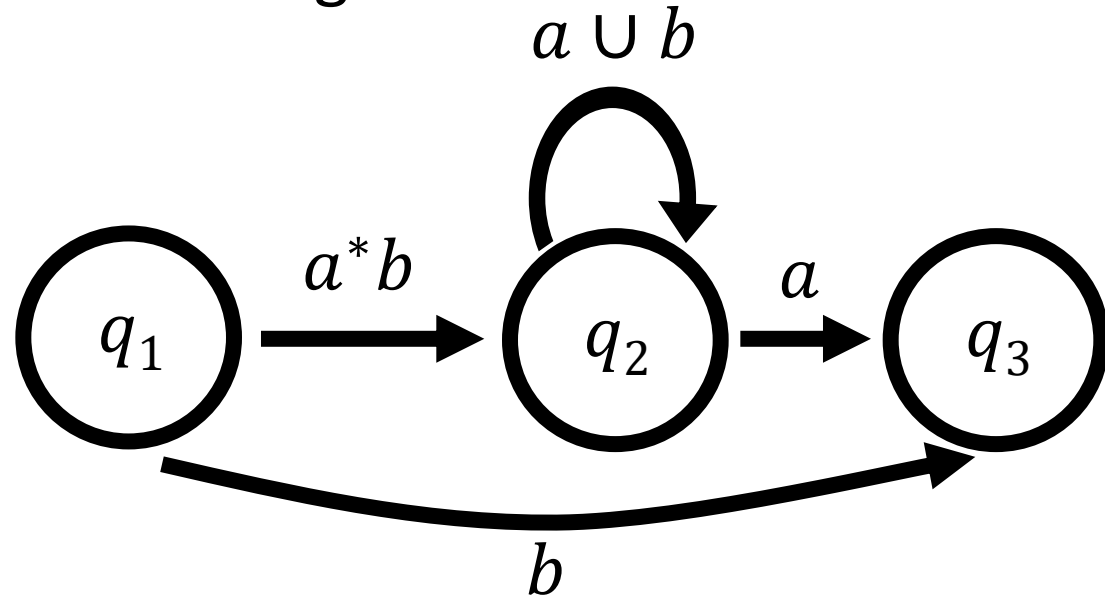
**Idea:** While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state

- a)  $a^*b(a \cup b)a$
- b)  $a^*b(a \cup b)^*a$**
- c)  $a^*b \cup (a \cup b) \cup a$
- d) None of the above



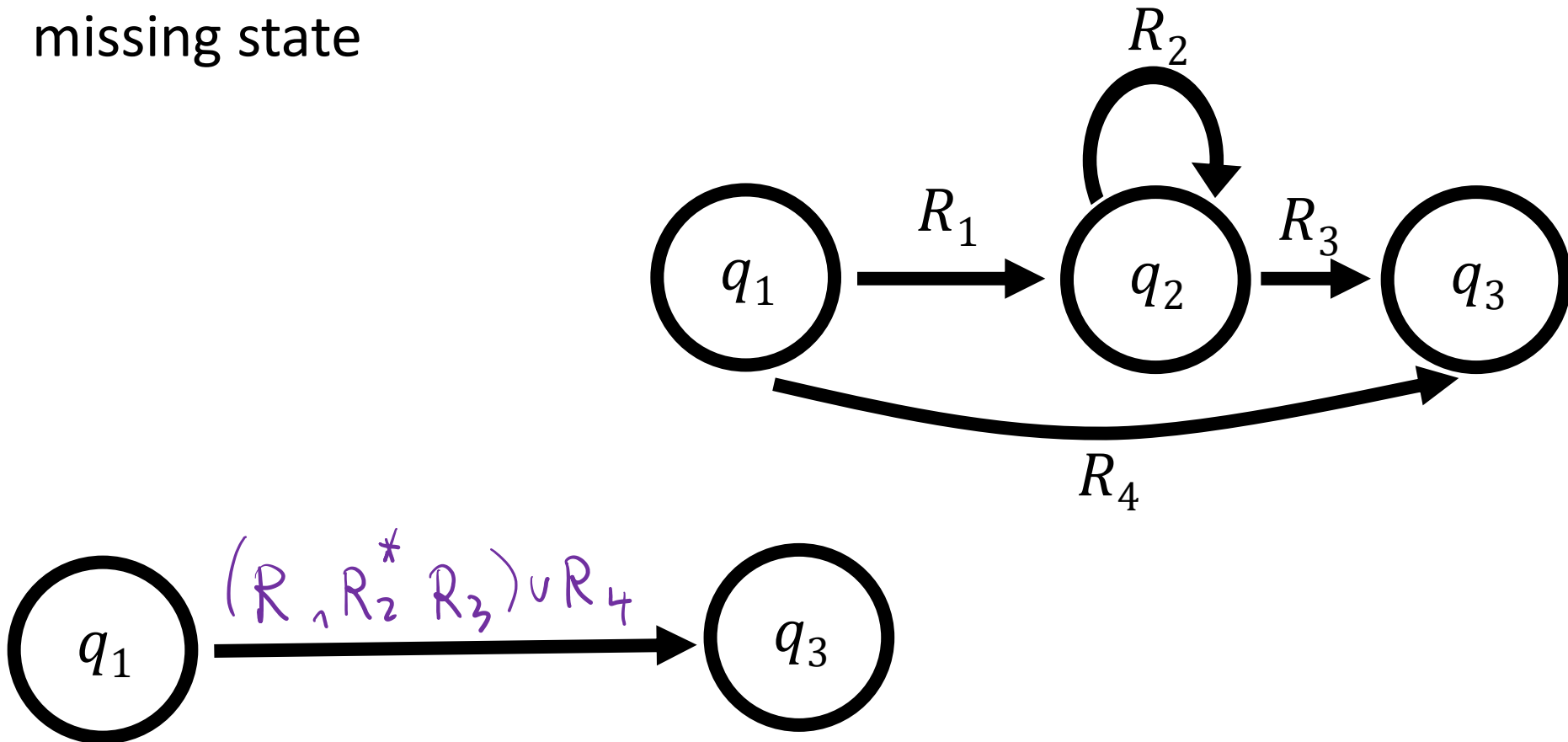
# GNFA $\rightarrow$ Regular expression

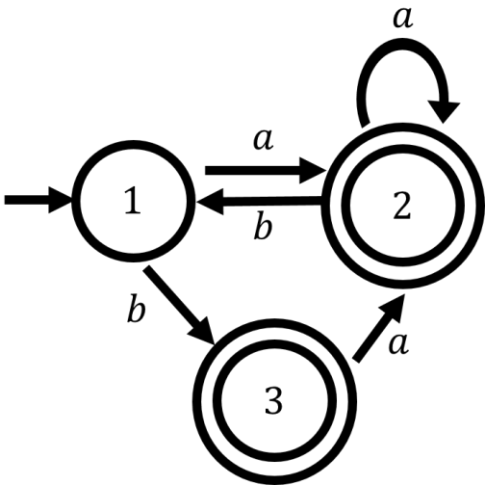
**Idea:** While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state



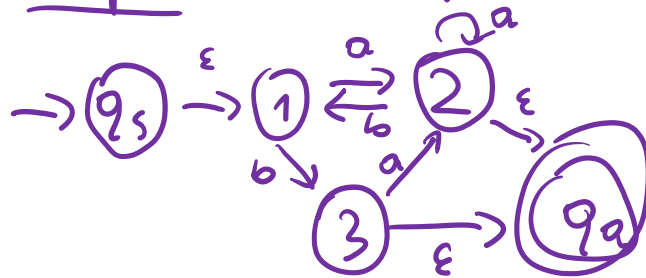
# GNFA $\rightarrow$ Regular expression

**Idea:** While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state

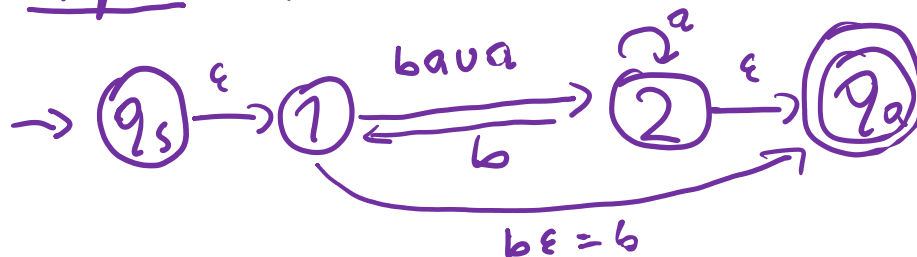




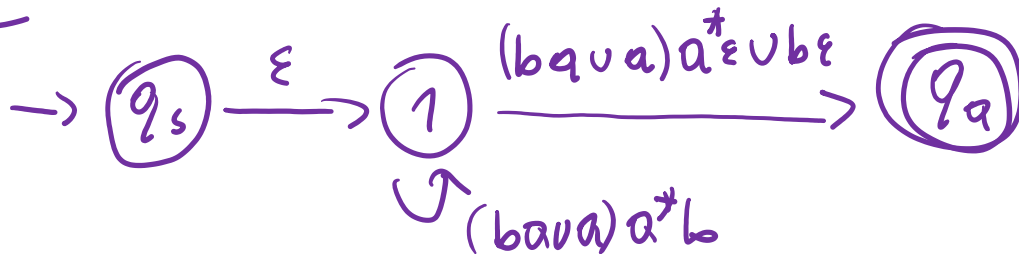
Step I: Add special vertices  $q_s$  and  $q_a$



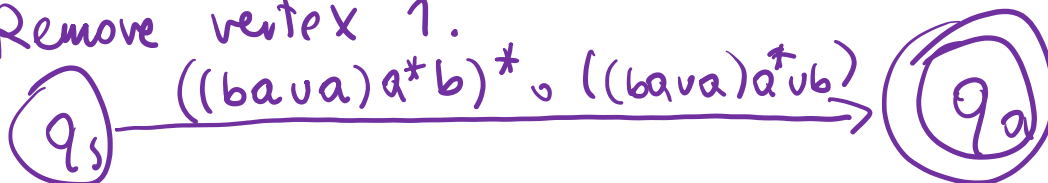
Step II: Remove vertex 3.



Step III Remove vertex 2.



Step IV: Remove vertex 1.



# Limitations of Finite Automata

# Motivating Questions

- We've seen techniques for showing that languages are regular
  
- How can we tell if we've found the smallest DFA recognizing a language?
- Are all languages regular? How can we prove that a language is not regular?