BU CS 332 – Theory of Computation

https://forms.gle/xcJwrTwrLQuitQrj9

Lecture 6:

- Regexes = NFAs
- Limitations of Finite Automata

Reading:

Sipser Ch 1.3

“Myhill-Nerode” note

Mark Bun

February 10, 2025

Regular Expressions – Syntax

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

1. ε, $\emptyset$, 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 $\quad$ $ab^*c \cup (a^*b)^*$ $\quad$ $\emptyset$

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. $L\left((R_1^*)\right) = (L(R_1))^*$

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

Syntactic Sugar

- For alphabet Σ, the regex Σ represents $L(\Sigma) = \Sigma$ $= \{a, b, c\}$

e.g. if $\Sigma = \{a, b, c\}$

then regular expression Σ stands for $(a \cup b \cup c)$

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

R^+ is interpreted as "one or more occurrence of a string generated by R"

Regexes in the Real World

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

```
$ grep '^xy*z' myfile
xyz
xyzde
xzz
xz
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
$\Leftrightarrow$ recognized by an NFA

Theorem 1: Every regular expression has an equivalent NFA

Theorem 2: Every NFA has an equivalent regular expression

Not regular: $\{a^n b^n \mid n \geq 0\}$

Regular: $\{a^n b^n \mid 0 \leq n \leq 2025\}$

Regular expression -> NFA

Theorem 1: Every regex has an equivalent NFA

Proof: Induction on size of a regex

Base cases:

	L(R)	NFA
$R = \emptyset$	ϕ	
$R = \varepsilon$	$\{\varepsilon\}$	
$R = a$	$\{a\}$	

Regular expression → NFA

$(a \cup b)^* \circ (c \cup b)$

Theorem 1: Every regex has an equivalent NFA

Proof: Induction on size of a regex = # of symbols, i.e. $\phi, \varepsilon, a, (,), \circ, \cup, {*}$

WTS statement is true for regexes of size $k+1$

size = length

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 → NFA

Theorem 1: Every regex has an equivalent NFA

Proof: Induction on size of a regex

IH: Assume every regex of size $\le k$ has an equiv. NFA

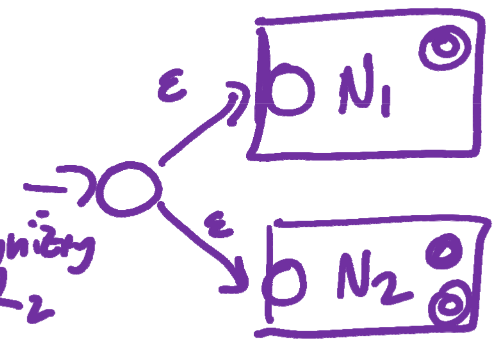
WTS if R is a regex of size $k+1$, then R has an equiv. NFA

Inductive step:

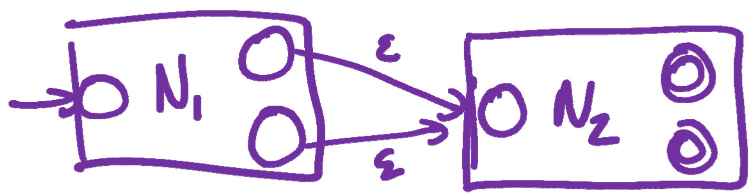
$$R = (R_1 \cup R_2)$$

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

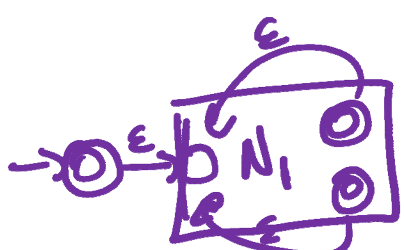
By IH, $\exists$ NFA N_1 recognizing R_1, N_2 recognizing R_2



$$R = (R_1 R_2)$$



$$R = (R_1^*)$$

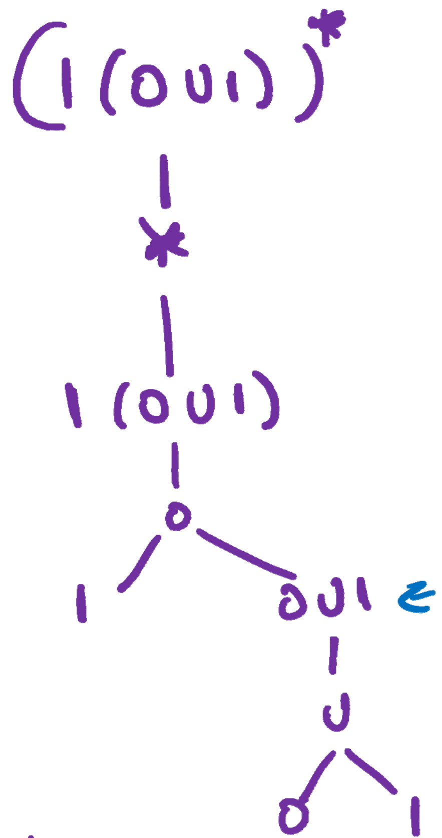


Example

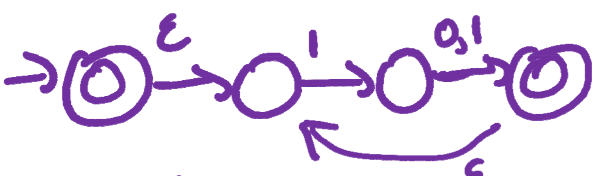
Convert $(1(0 \cup 1))^*$ to an NFA

e.g. 10 11 10 11   101010

= { even length strings w/ 1 in every odd position }



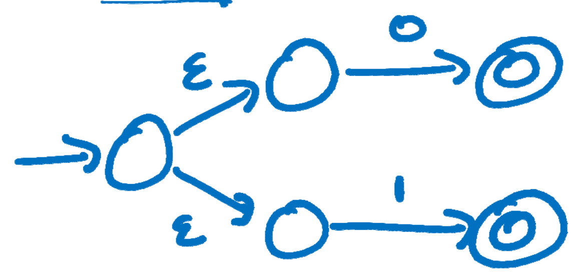
Simplify to:



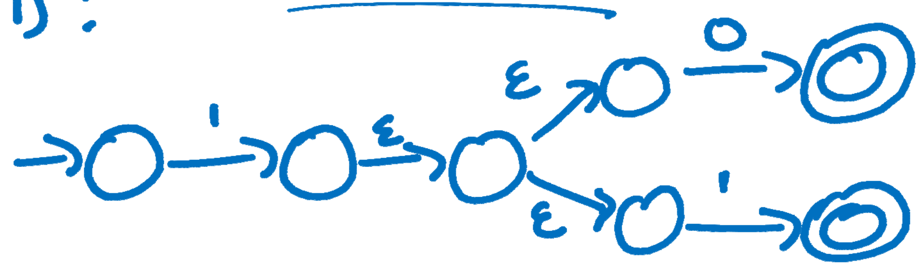
0 :

1 :

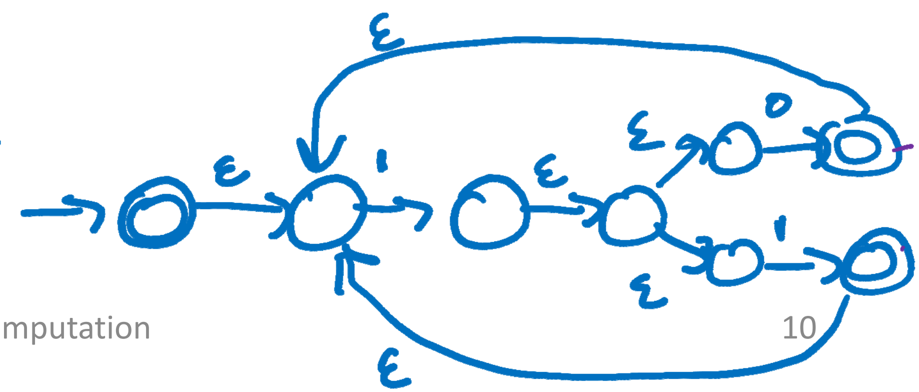
0 ∪ 1 :



1 ∘ (0∪1) :



(1 (0∪1))* :



Regular Expressions Describe Regular Languages

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

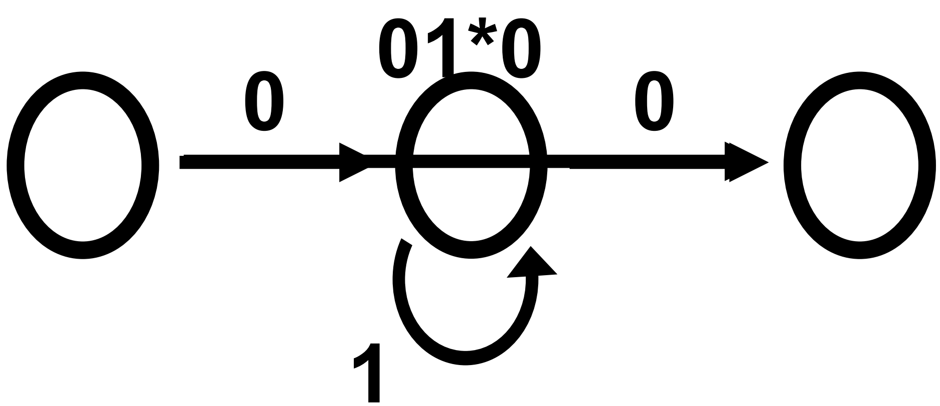
Theorem 1: Every regular expression has an equivalent NFA

Theorem 2: Every NFA has an equivalent regular expression

NFA → Regular expression

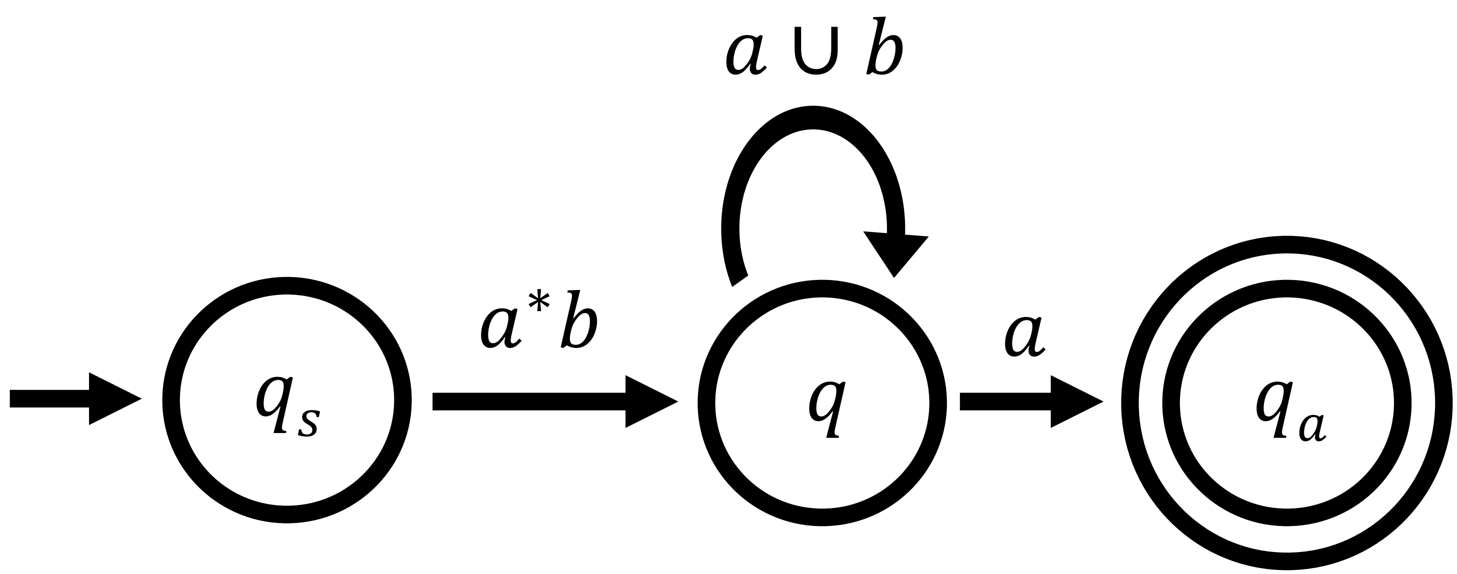
Theorem 2: Every NFA has an equivalent regex

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

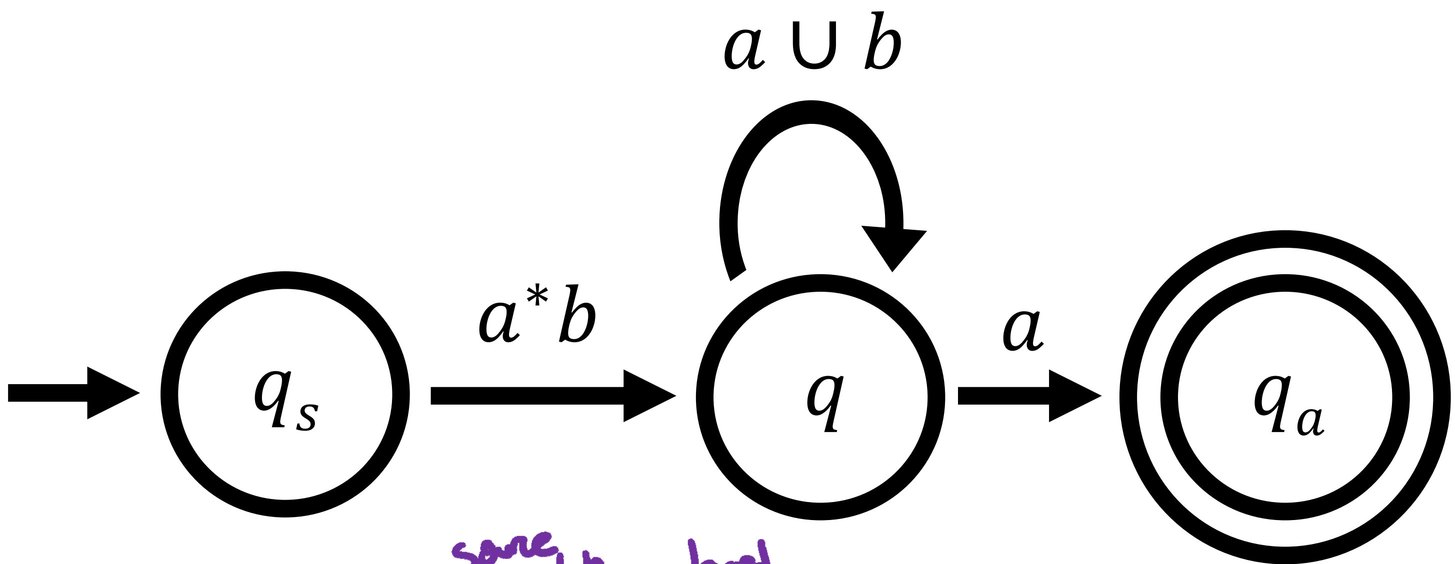


Generalized NFAs

- **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



source state
target state

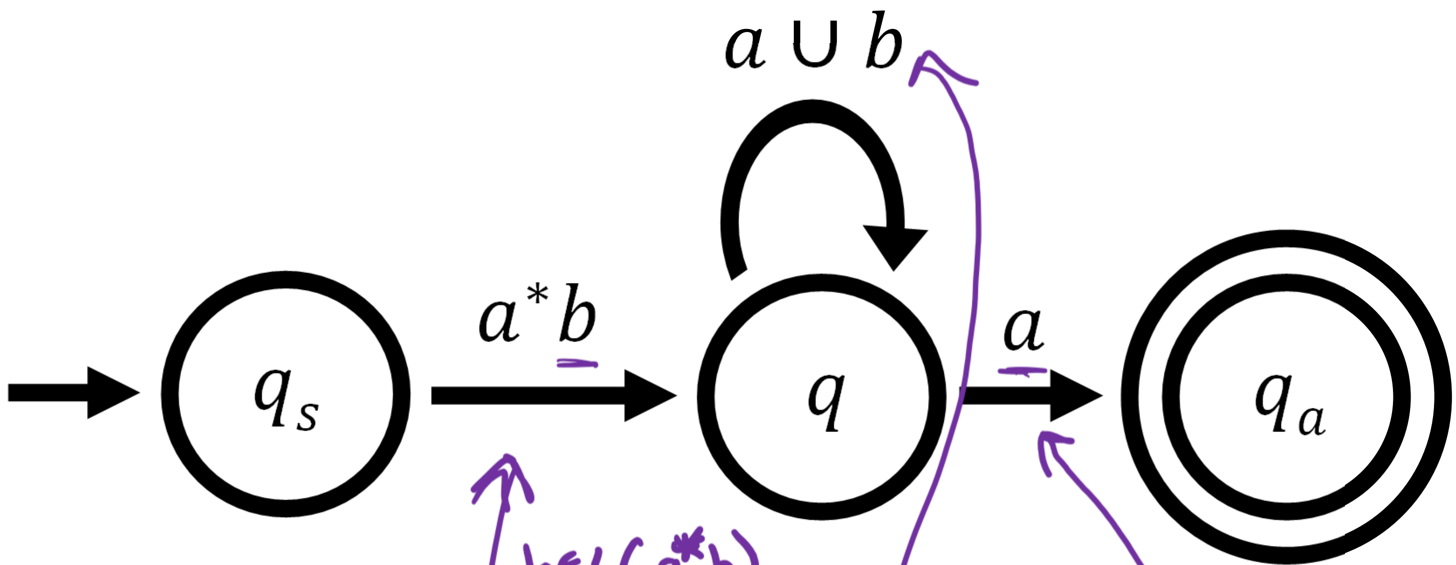
$$R(q_s, q) = a^* b$$

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

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

Which of these strings is accepted?

Which of the following strings is accepted by this GNFA?



$b \in L(a^*b)$

$b \in L(a \cup b)$

$a \in L(a)$

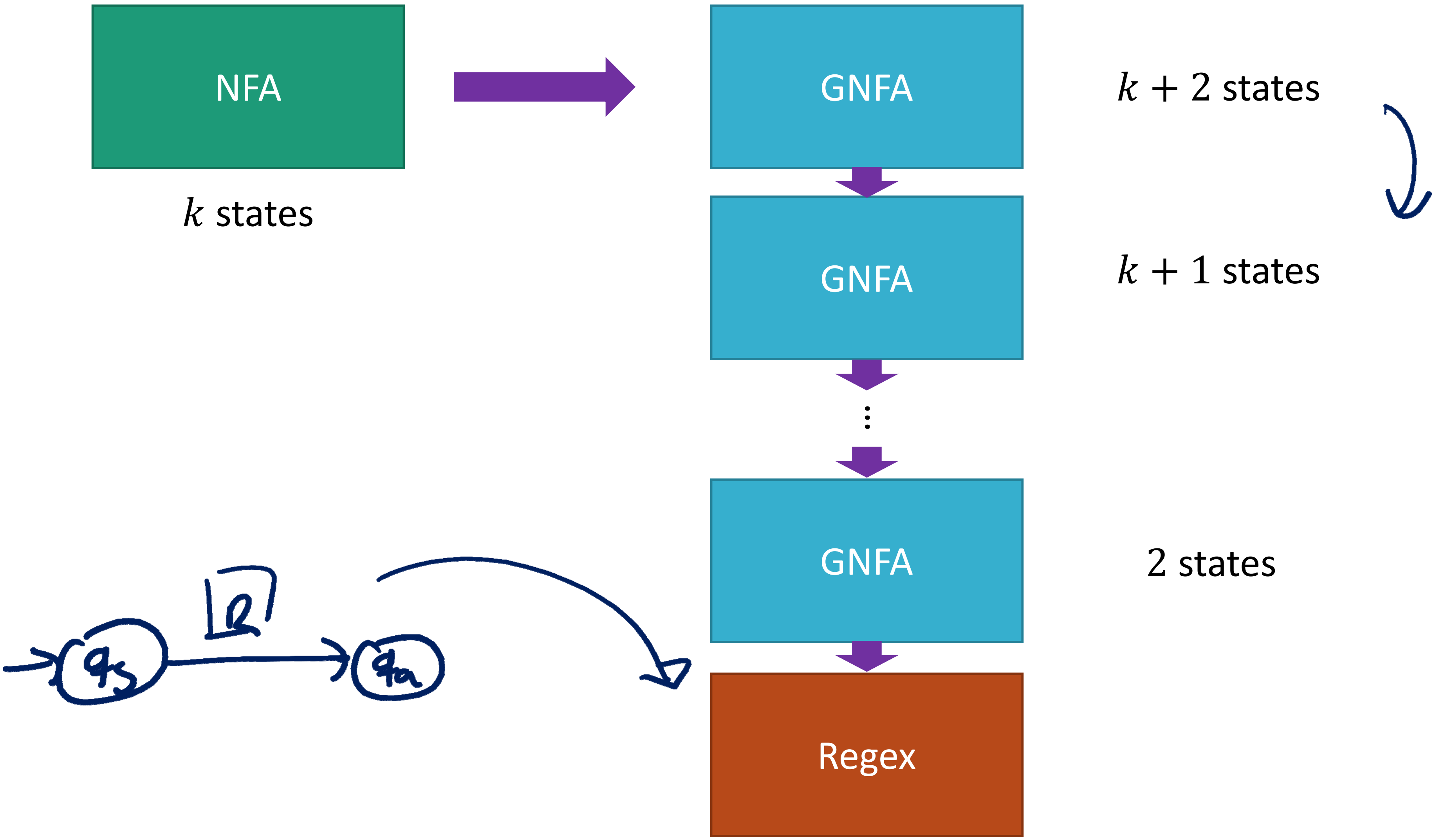
- a) ~~aaa~~
- b) ~~$aabb$~~
- c) ~~bbb~~
- $\boxed{\text{d)}}$ bba

$q_s \xrightarrow{b} q \xrightarrow{b} q \xrightarrow{a} q_a$

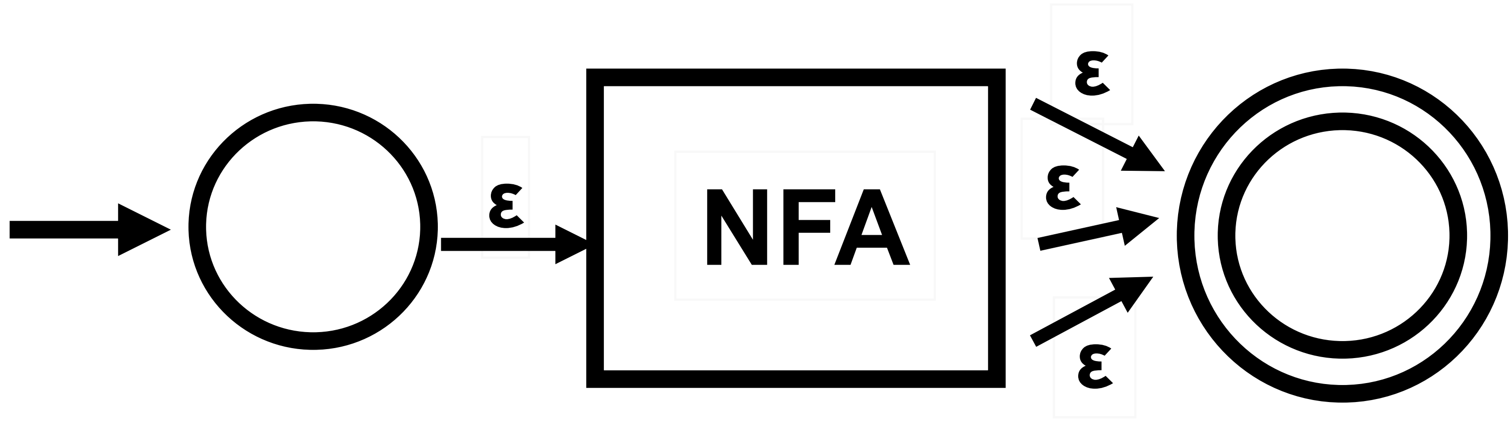
ba   a

abba

NFA → Regular expression



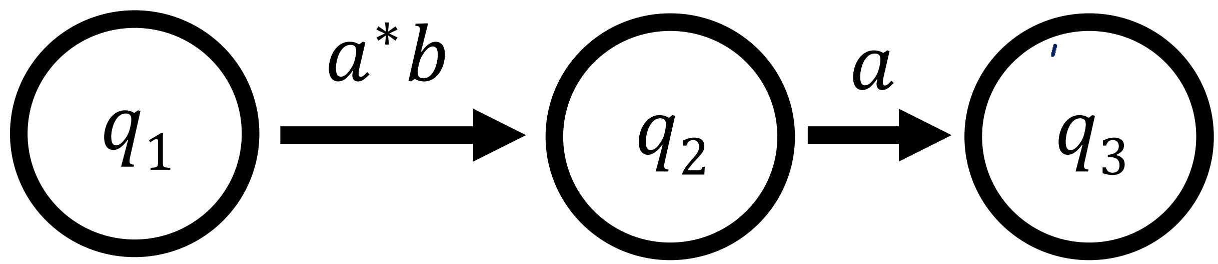
NFA → GNFA



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

GNFA → Regular expression

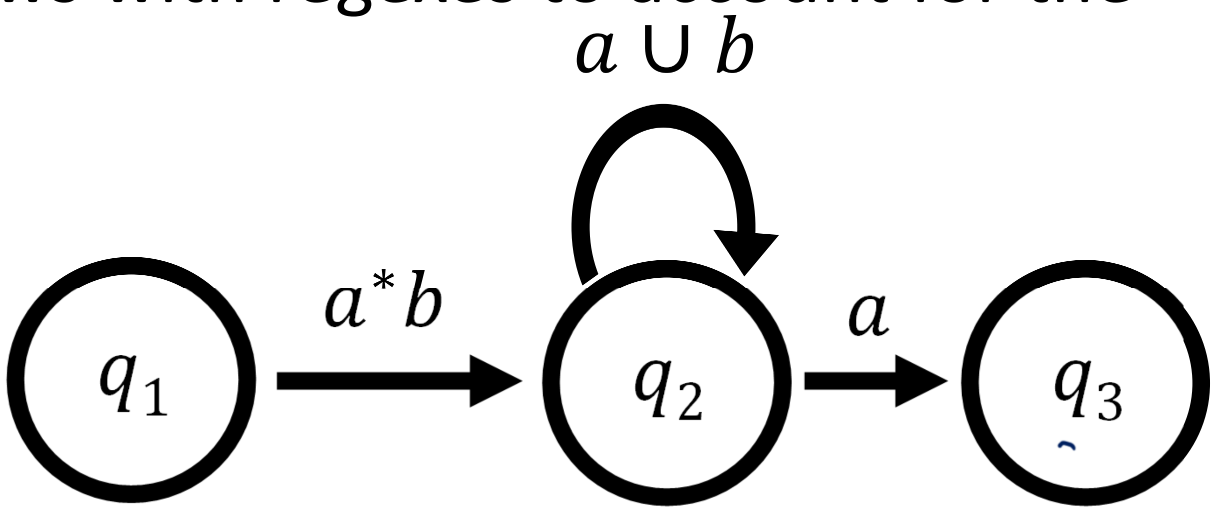
Idea: While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state

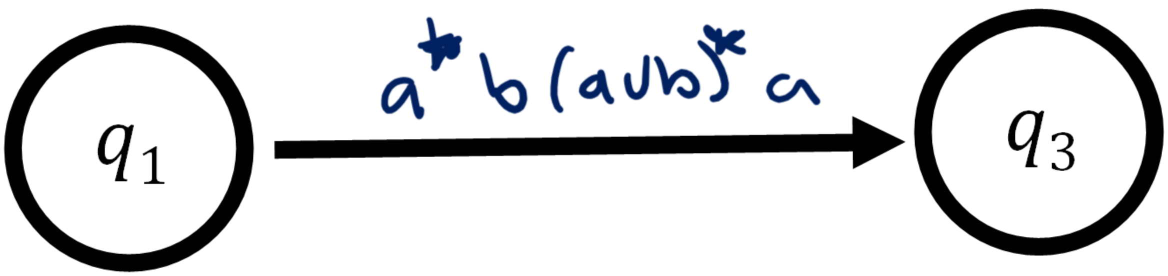


GNFA → Regular expression

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

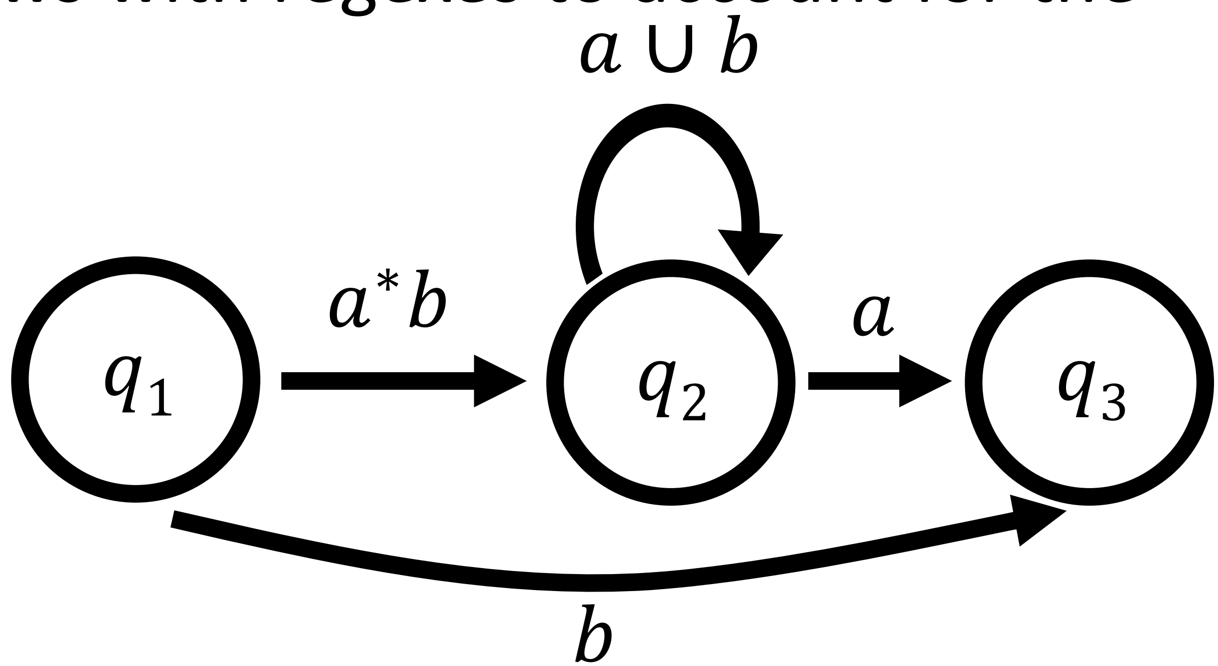
a) $a^*b(a \cup b)a$
b) $a^*b(a \cup b)^*a$
c) $a^*b \cup (a \cup b) \cup a$
d) None of the above

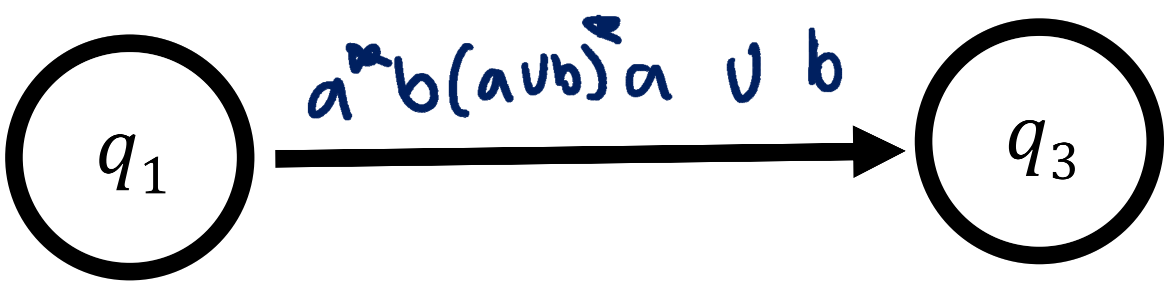




GNFA → Regular expression

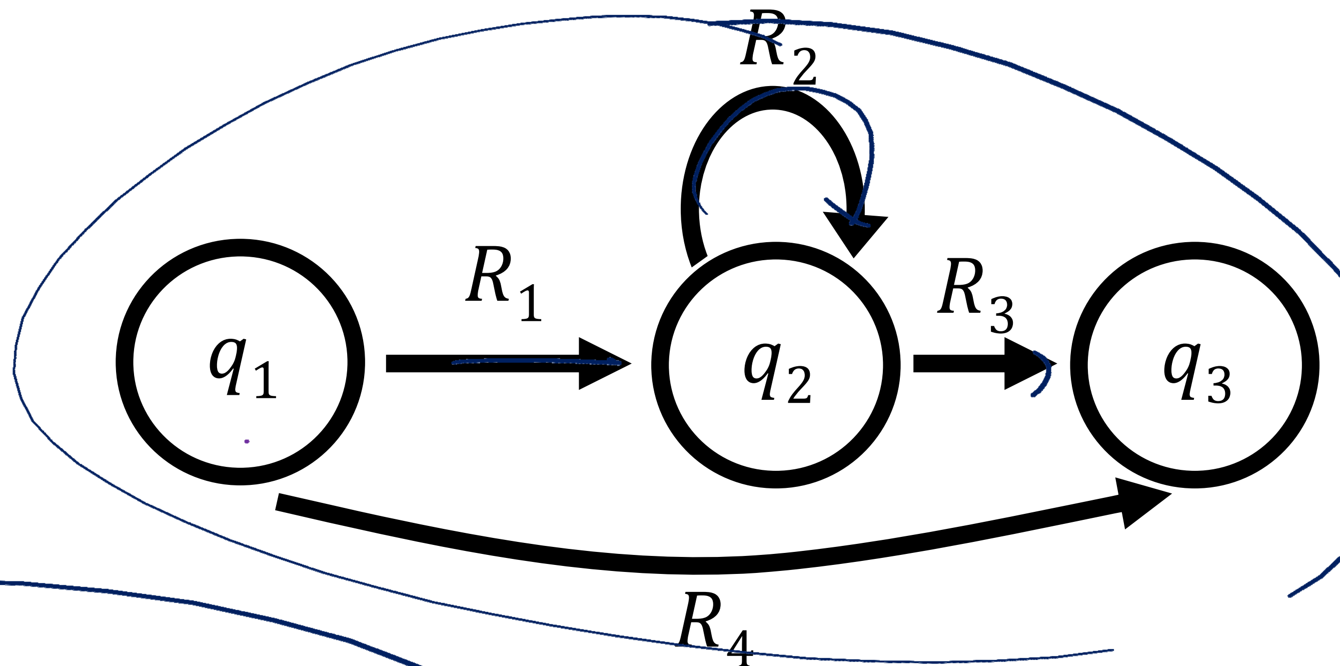
Idea: While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state

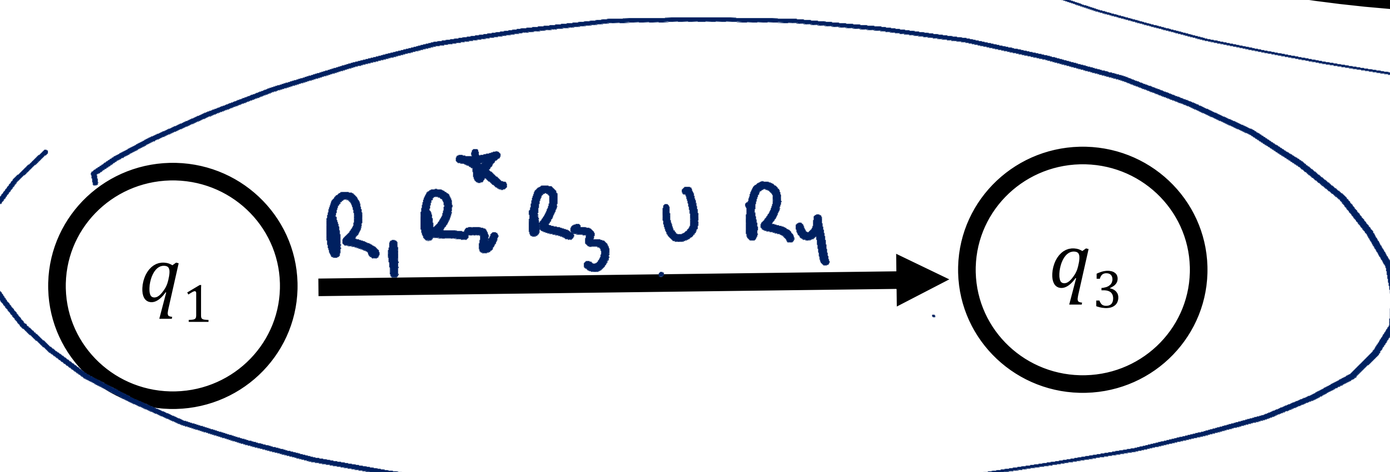


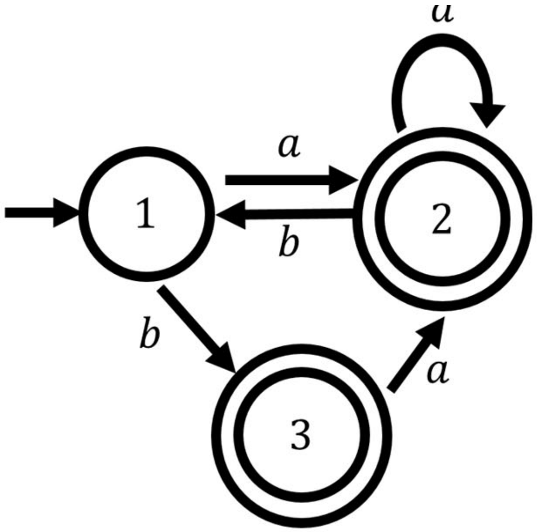


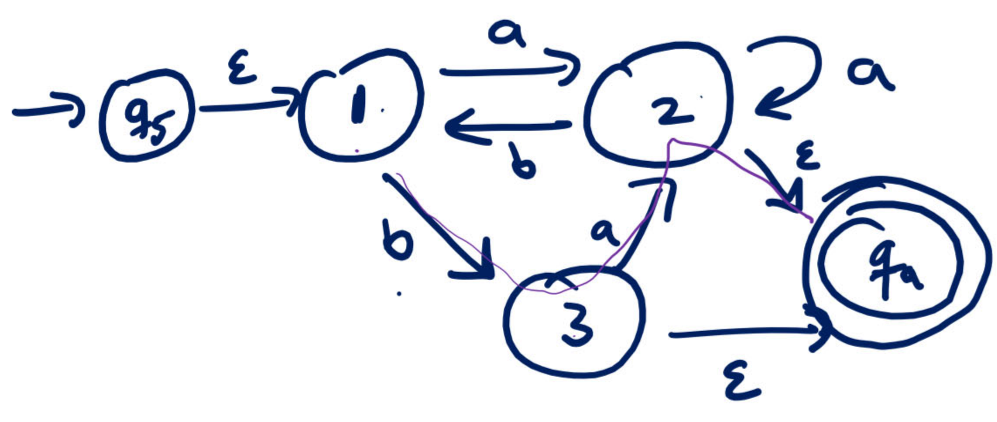
GNFA -> Regular expression

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



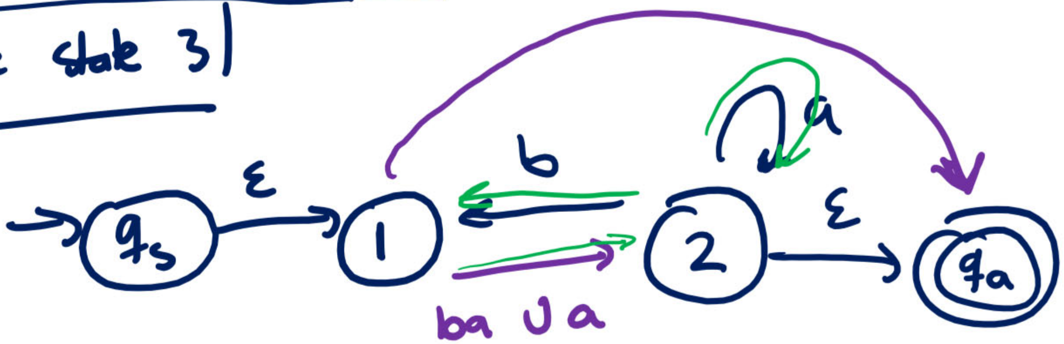






$b\varepsilon \; (= b)$

Remove state 3



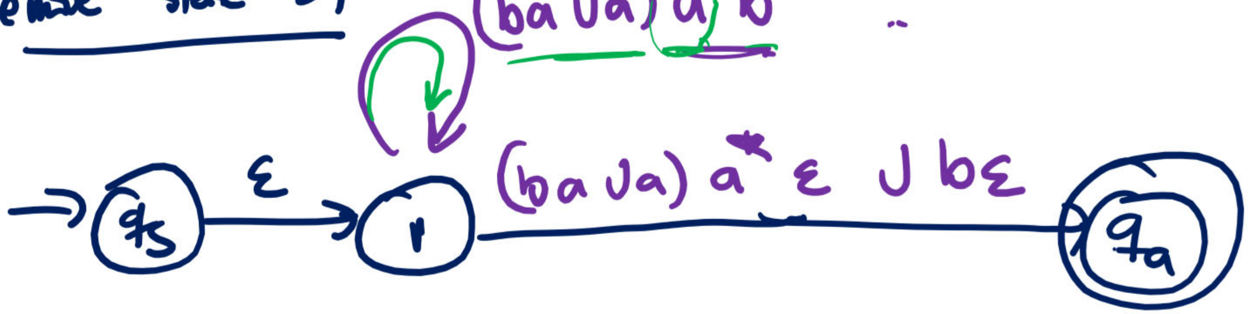
Remove state 3

$$\varepsilon\left(\left(ba \cup a\right)a^*b\right)^* \circ \left(\left(ba \cup a\right)a^*\varepsilon \cup b\varepsilon\right)$$

$$\equiv \left(\left(ba \cup a\right)a^*b\right)^* \circ \left(\left(ba \cup a\right)a^* \cup b\right)$$

Remove state 2

$(ba \cup a)\,a^*\,b$



Limitations of Finite Automata

Motivating Questions

- We've seen techniques for showing that languages are regular
  - Construct a DFA
  - Construct an NFA
  - Construct a regex
  - Use closure properties

- 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?