BU CS 332 – Theory of Computation

https://forms.gle/5sTNDCU1QtEemHHM7

Lecture 6:

- Regexes = NFAs
- Limitations of Finite Automata

Reading:

Sipser Ch 1.3

“Myhill-Nerode” note

Mark Bun

February 7, 2024

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

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

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

Syntactic Sugar

$0 \cup 1$

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

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

same as R^*, but not including ε

$L(R^+) =$ strings obtained by concatenating one or more strings from $L(R)$

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 DFAs

$\Leftrightarrow$ recognized by NFAs

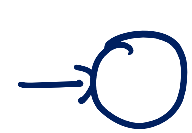
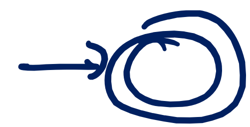
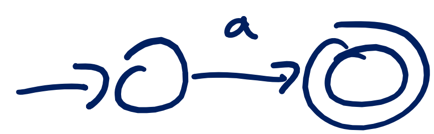
Theorem 1: Every regular expression has an equivalent NFA

Theorem 2: Every NFA has an equivalent regular expression

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

Theorem 1: Every regex has an equivalent NFA

Proof: Induction on size of a regex

IH $\Rightarrow$ every regex of size $k+1$ has an equiv. NFA.

"# of symbols in regex, i.e. # of $\emptyset, \varepsilon, a, (,), \cup, \circ, *$

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

$(R_1 \cup R_2)$

Regular expression -> NFA

Theorem 1: Every regex has an equivalent NFA

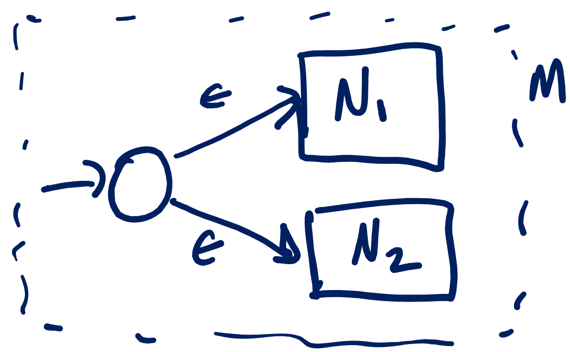
Proof: Induction on size of a regex

Assume every regex of size $\leq k$ can be converted to an NFA

Let R be an arbitrary regex of size $k+1$.

Inductive step:

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

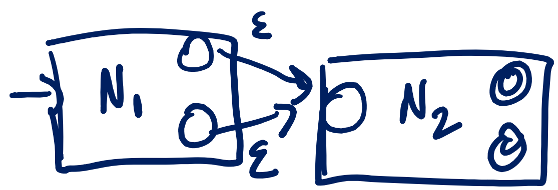


Let N_1 recognize $L(R_1)$

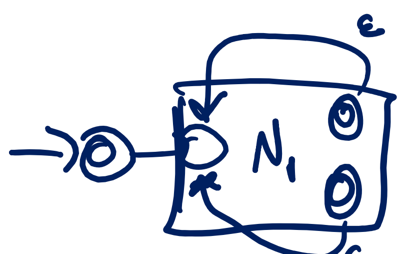
N_2 " $L(R_2)$

$\Rightarrow M$ recognizes $L(R) = L(R_1) \cup L(R_2)$

$$R = (R_1 R_2)$$

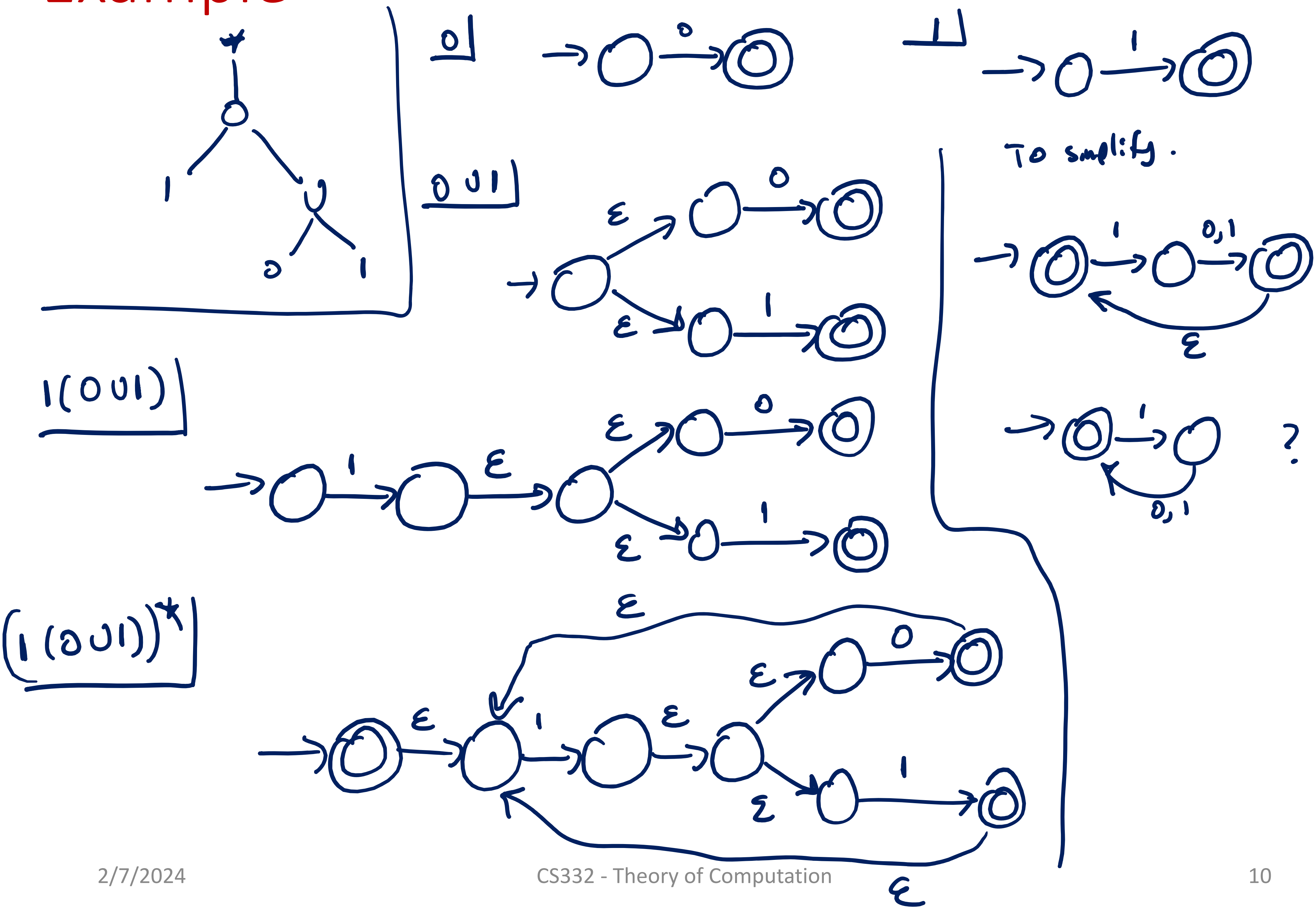


$$R = (R_1^*)$$



Example

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



Regular Expressions Describe Regular Languages

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

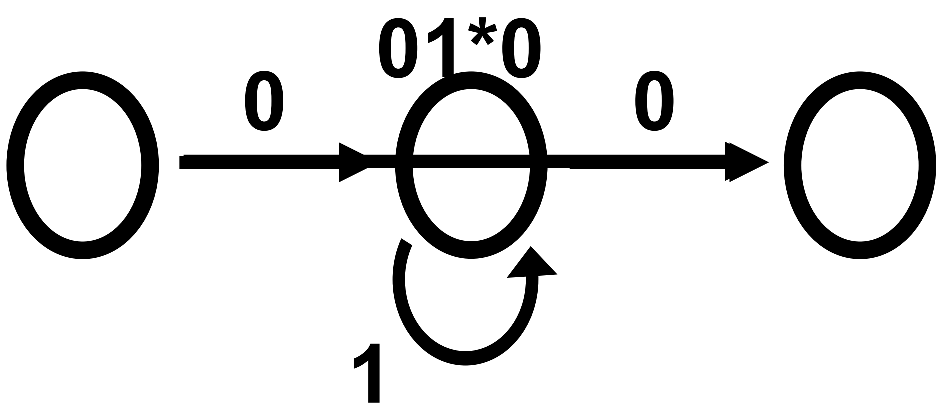
Theorem 1: Every regular expression has an equivalent NFA

Theorem 2: Every NFA has an equivalent regular expression

NFA -> Regular expression

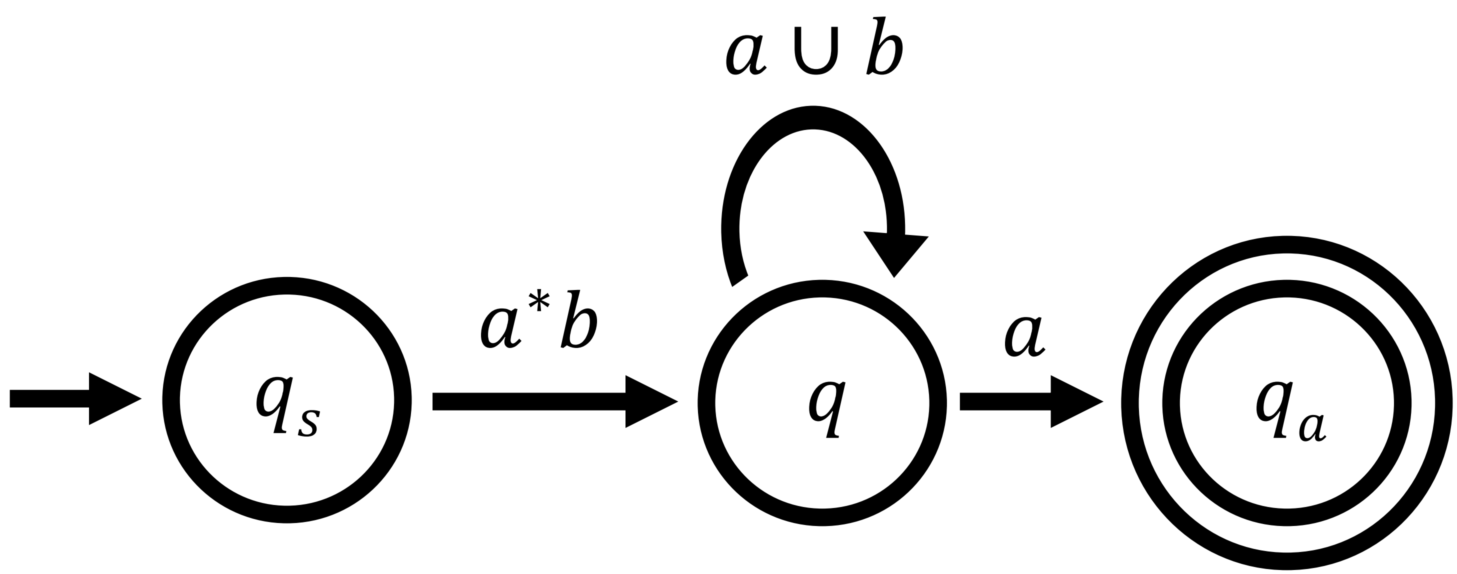
Theorem 2: Every NFA has an equivalent regex

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

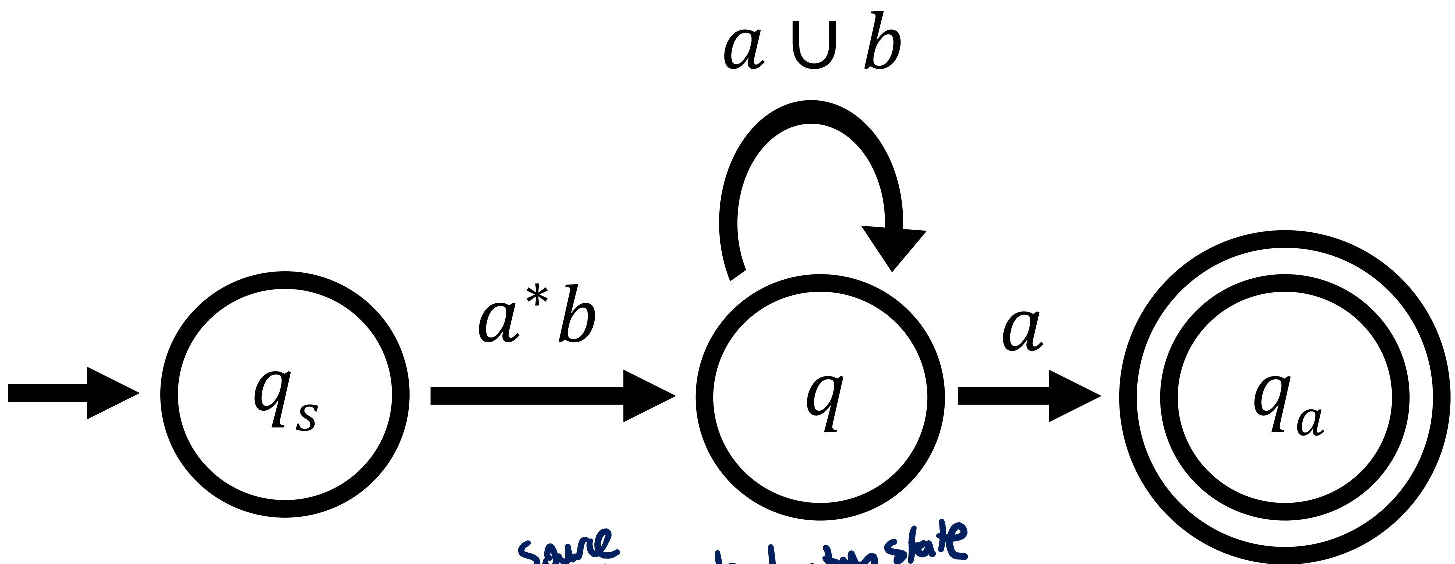


Generalized NFAs (GNFAs)

- **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

destination state

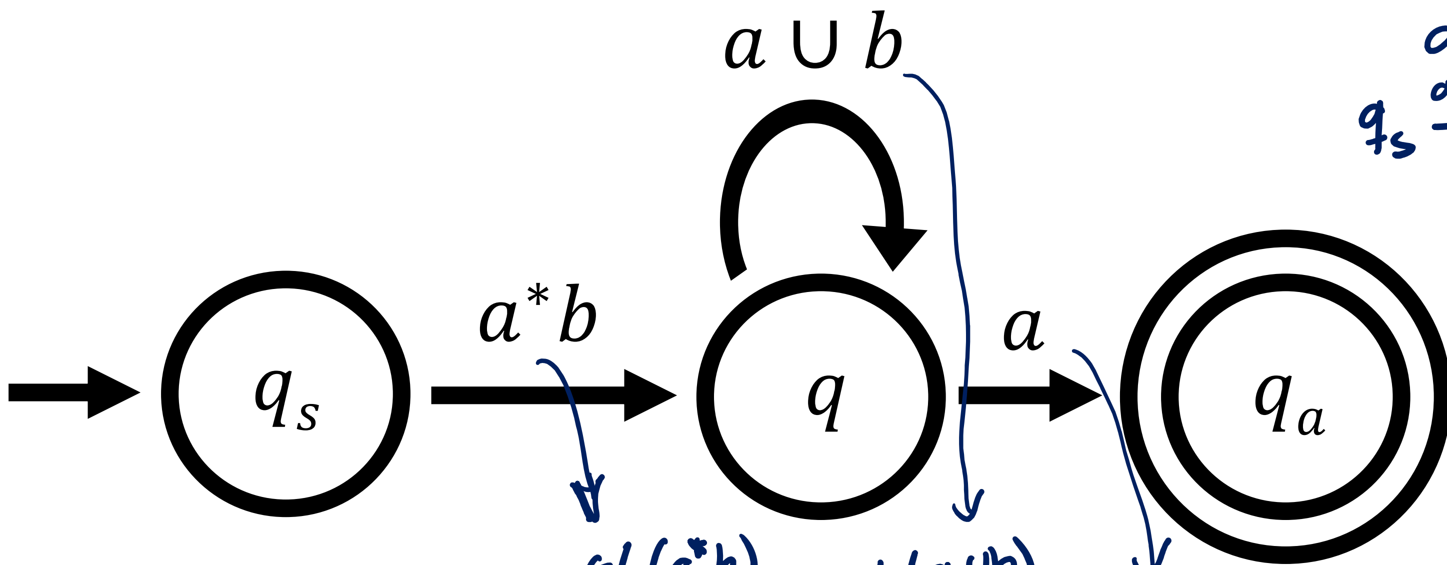
$R(q_s, q) = a^*b$ ← regex labeling transition from source → dest.

$R(q_a, q) = \phi$

$R(q, q_s) = \phi$

Which of these strings is accepted?

Which of the following strings is accepted by this GNFA?



aaba is accept

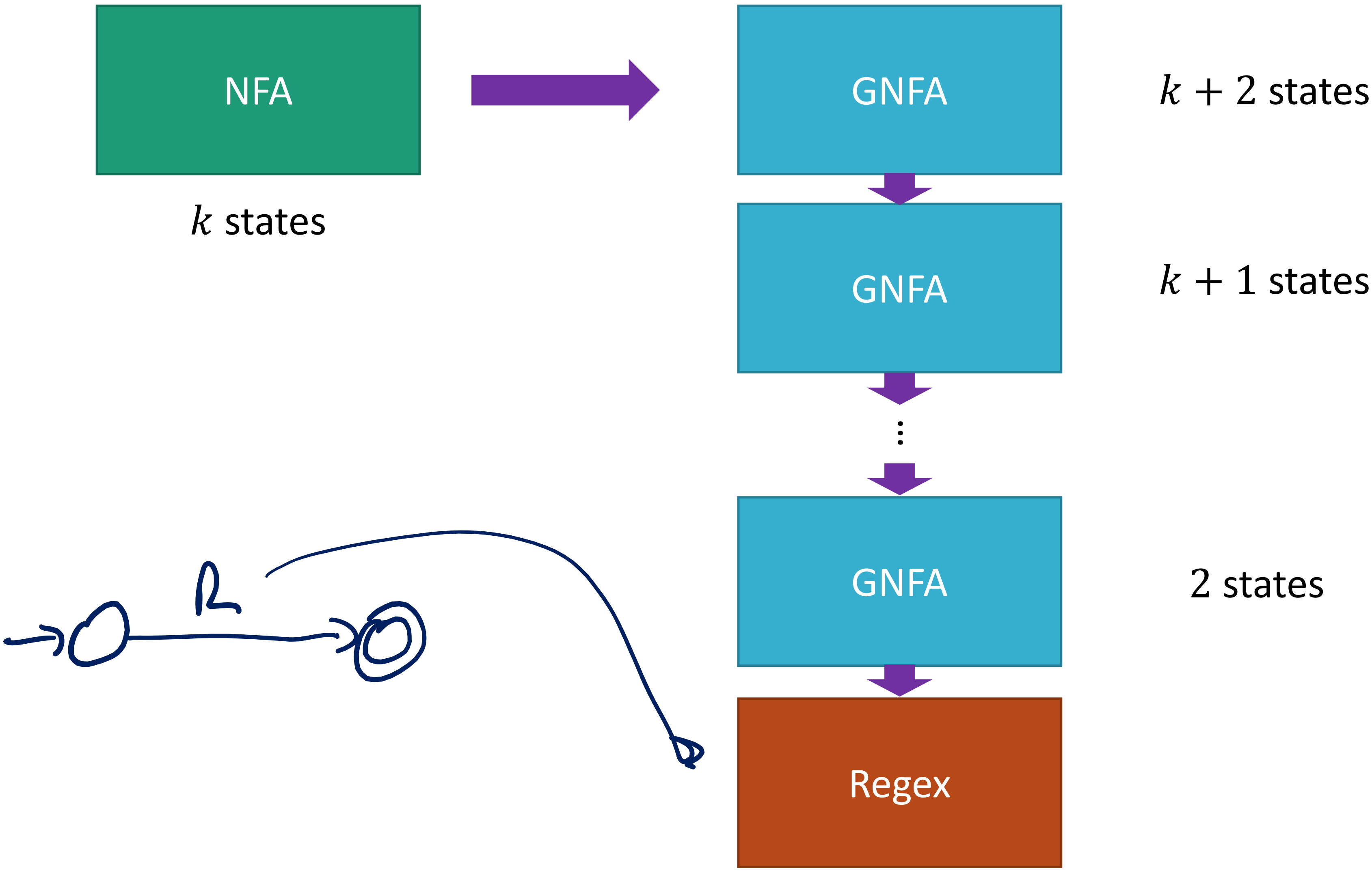
$q_s \xrightarrow{aab} q \xrightarrow{a} q_a$

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

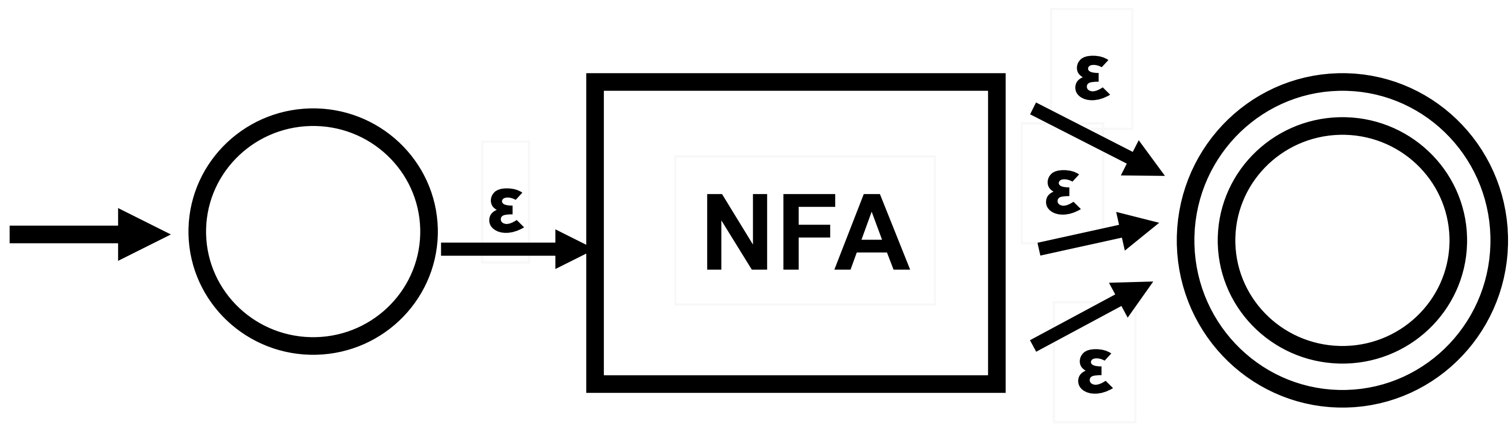
$q_s \xrightarrow{b} q \xrightarrow{b} q \xrightarrow{a} q_a$

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

NFA -> Regular expression



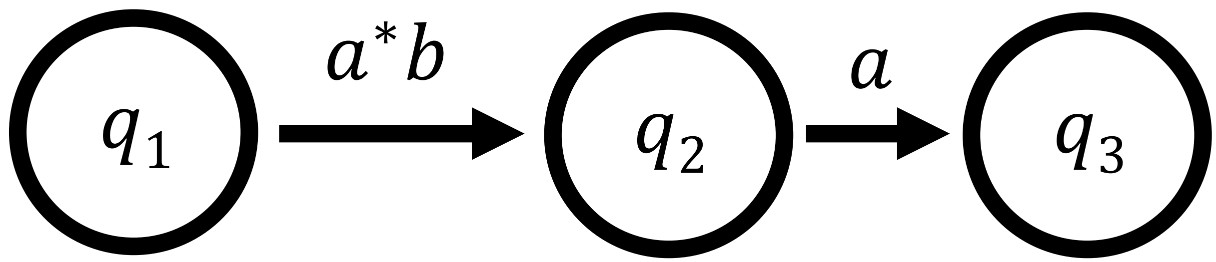
NFA -> GNFA



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

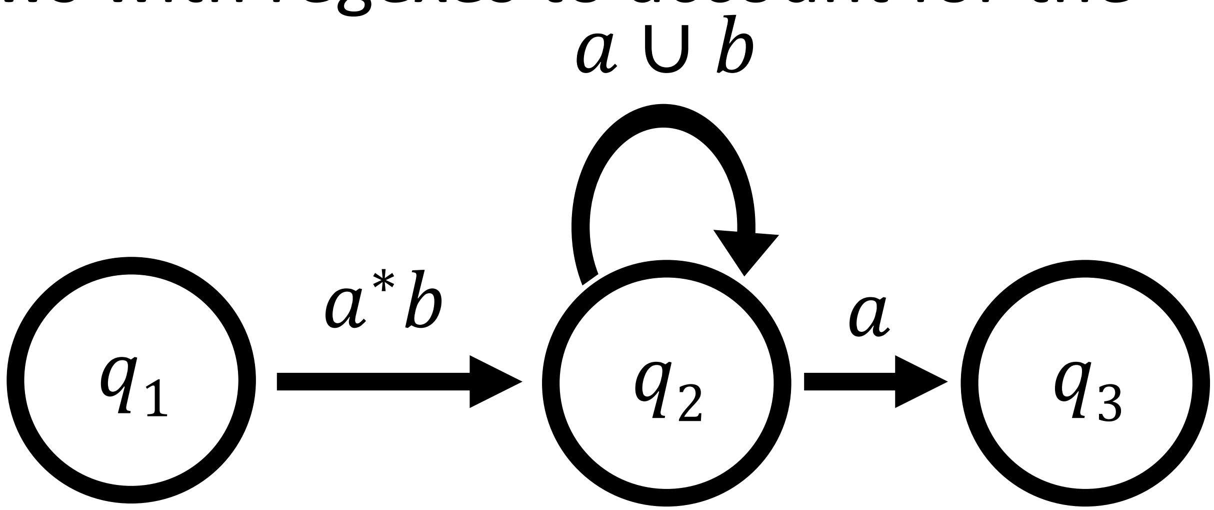
GNFA -> Regular expression

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

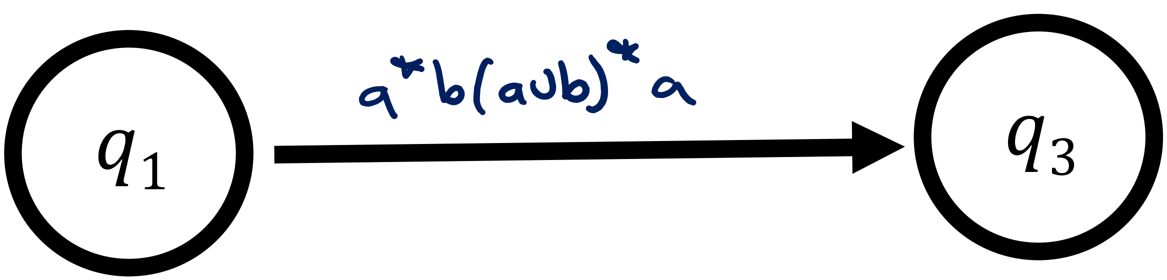


GNFA -> Regular expression

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

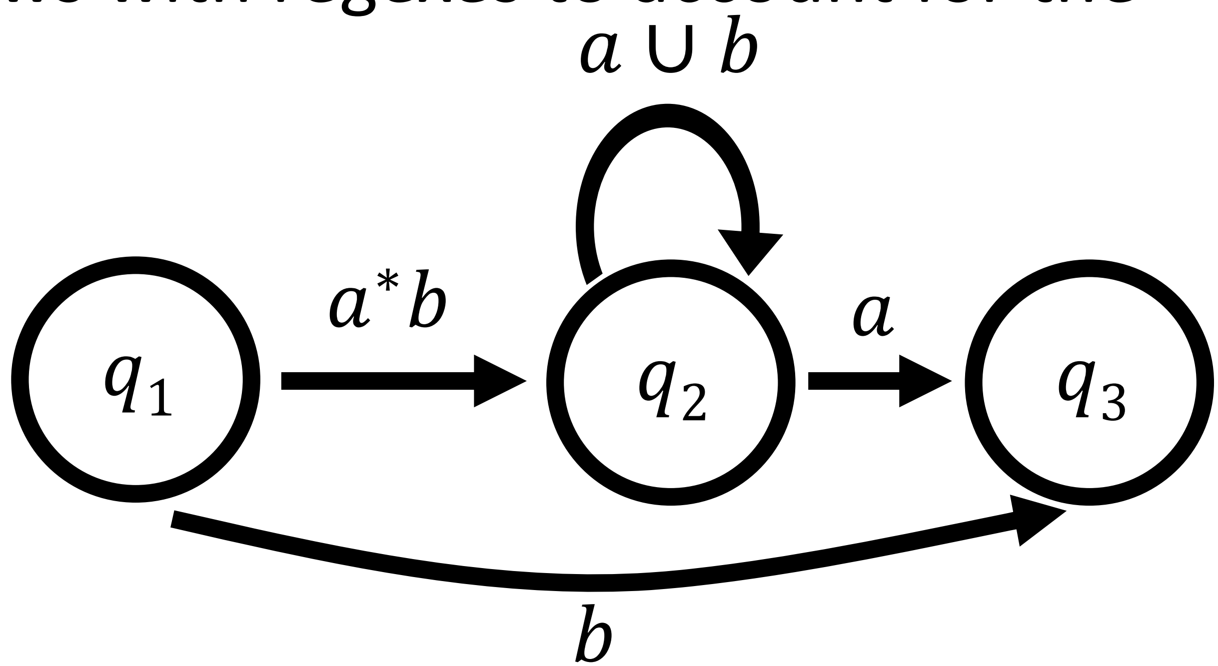


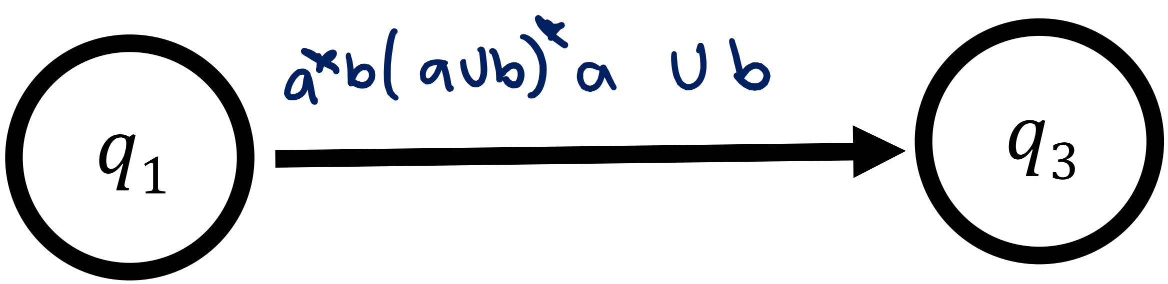
a) $a^*b(a \cup b)a$
b) $a^*b(a \cup b)^*a$
c) $a^*b \cup (a \cup b) \cup a$
d) None of the above



GNFA -> Regular expression

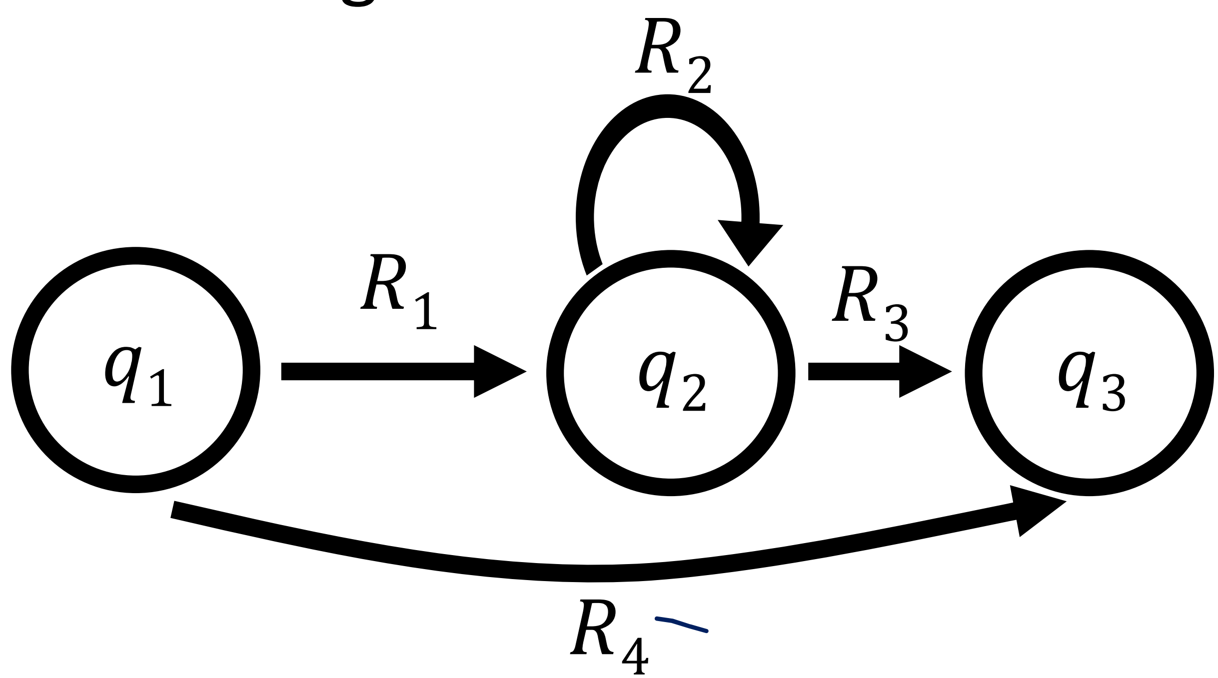
Idea: While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state

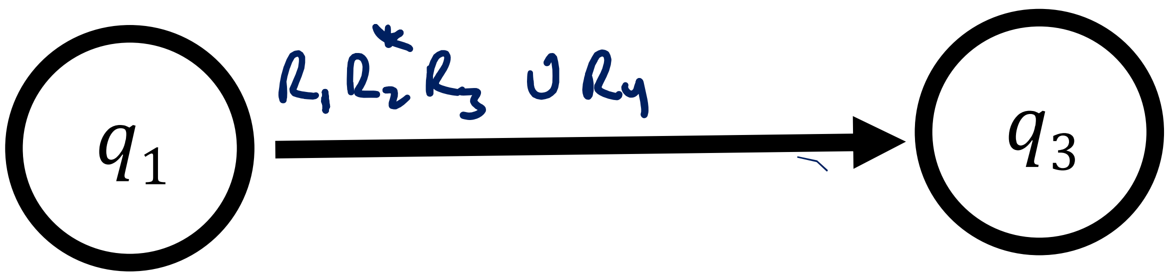


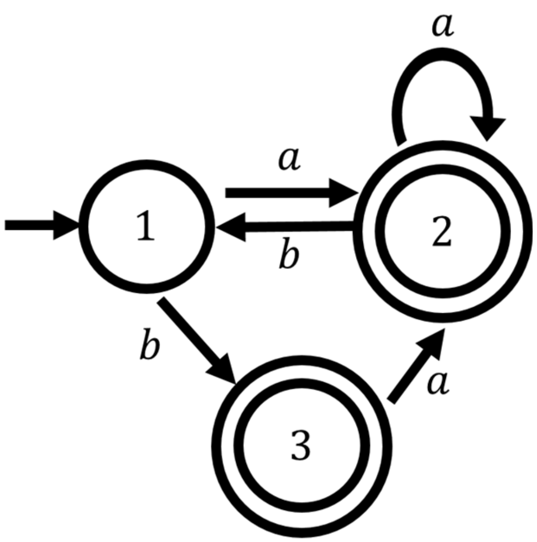


GNFA -> Regular expression

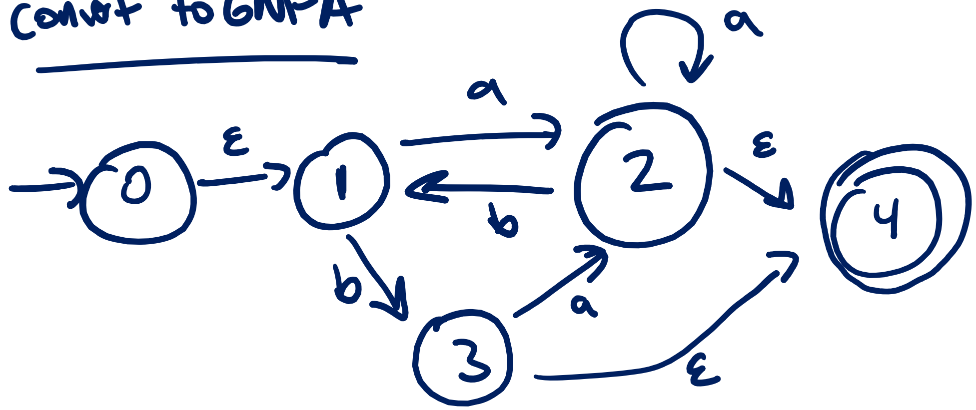
Idea: While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state



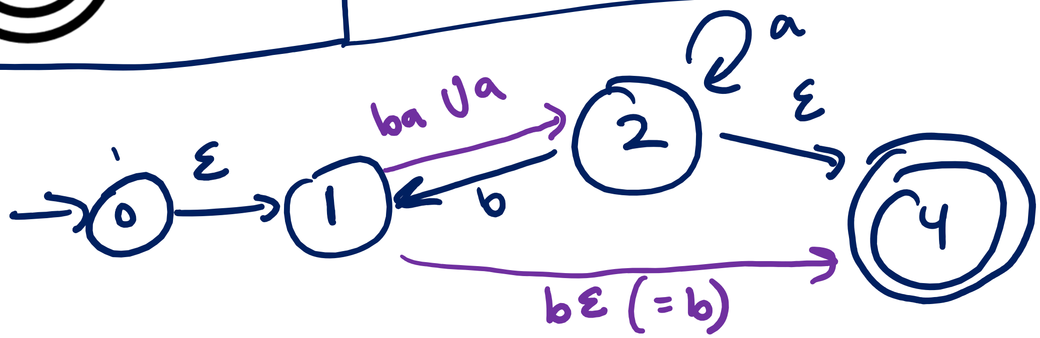




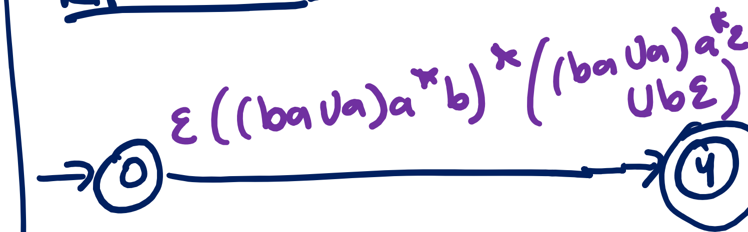
Convert to GNFA



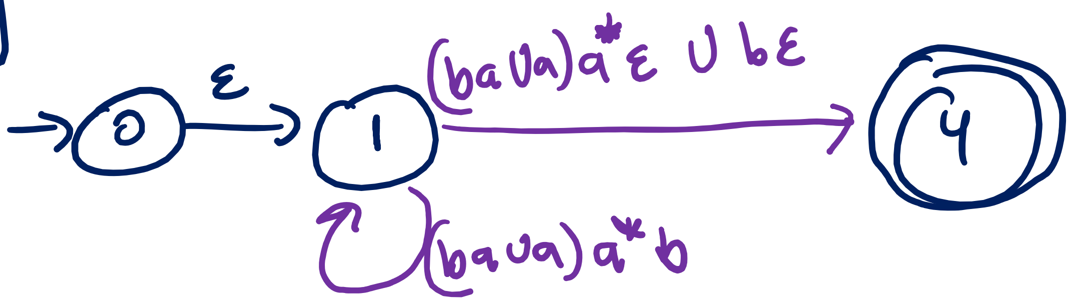
Rip out 3



Rip out 1



Rip out 2

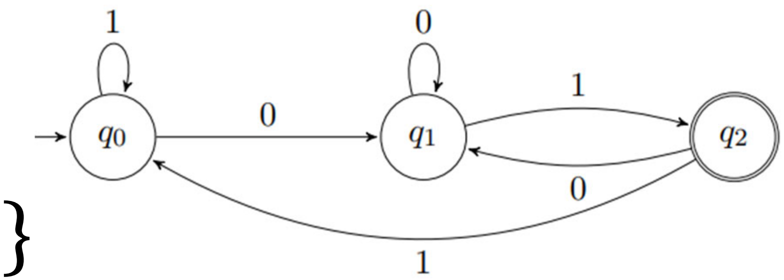


Limitations of Finite Automata

Motivating Questions

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



An Example

$A = \{ w \in \{0,1\}^* \mid w \text{ ends with } 01 \}$

Claim: *Every* DFA recognizing A needs at least 3 states

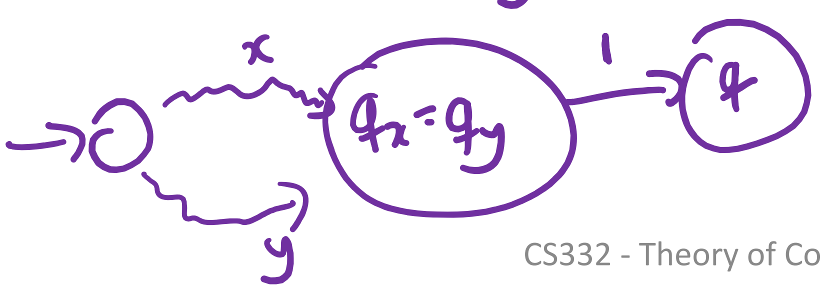
Proof: Let M be any DFA recognizing A. Consider running M on each of $x = \varepsilon, y = 0, w = 01$

Let q_x = state M reaches after reading x
q_y = " y
q_w = " w

Goal: Prove that q_x, q_y, q_z are all distinct.

Claim: $q_x \neq q_w$ and $q_y \neq q_w$ Why? q_w is an accept state
q_x, q_y are reject states

Claim: $q_x \neq q_y$
Proof: Suppose FTSOC $q_x = q_y$



M on input $x1 = 1$ must reject

M on input $y1 = 01$ must accept

$\ast$ because q can't be both an accept & reject state