BU CS 332 – Theory of Computation

Lecture 6:

- Regexes = NFAs
- Non-regular languages

Reading:
Sipser Ch 1.3
"Myhill-Nerode" note
Sipser Ch 1.4 (optional)

Mark Bun

February 10, 2021

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

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

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

Regular expression -> NFA

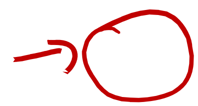
Theorem 1: Every regex has an equivalent NFA

Proof: Induction on size of a regex

Base cases:

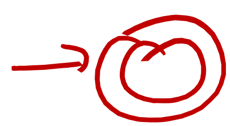
recognize

$R = \emptyset$



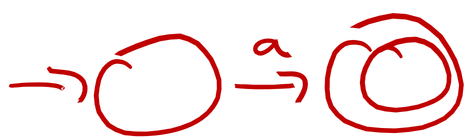
ϕ

$R = \varepsilon$



$\{\epsilon\}$

$R = a$



$\{a\}$

Regular expression -> NFA

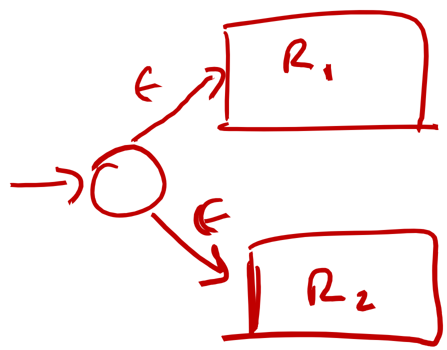
Theorem 1: Every regex has an equivalent NFA

Proof: Induction on size of a regex

recognize

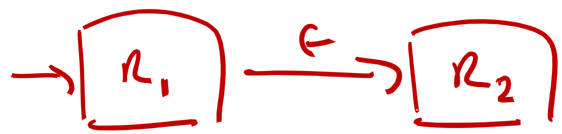
Inductive step:

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



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

$$R = (R_1 R_2)$$



$L(R_1) \circ L(R_2)$

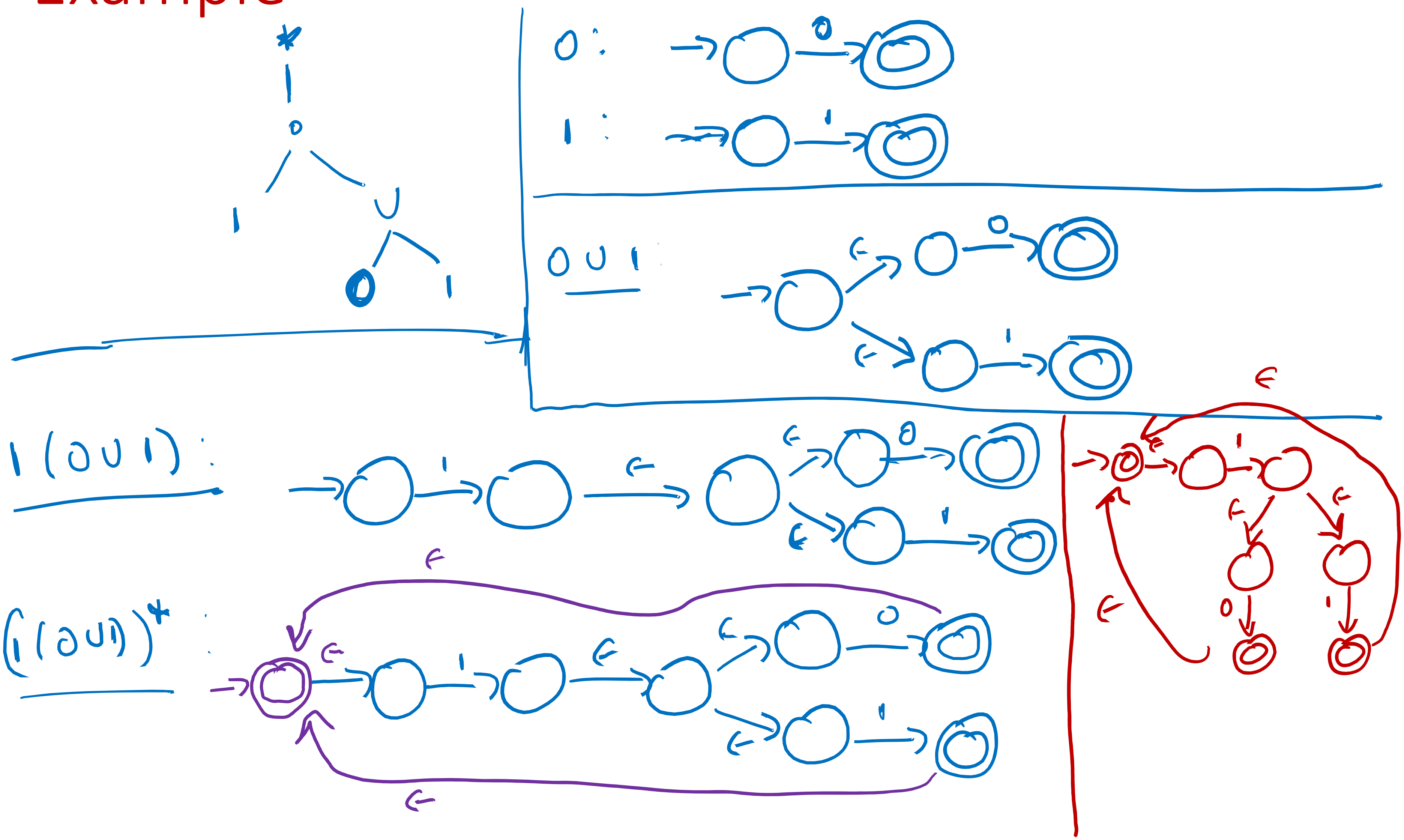
$$R = (R_1^*)$$

$(L(R_1))^*$

Example

even length strings s.t. every odd pos. = 1

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

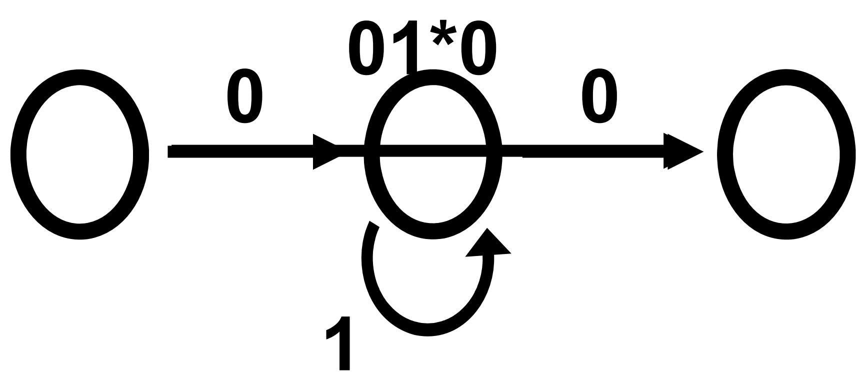
Theorem 1: Every regular expression has an equivalent NFA

Theorem 2: Every NFA has an equivalent regular expression

NFA -> Regular expression

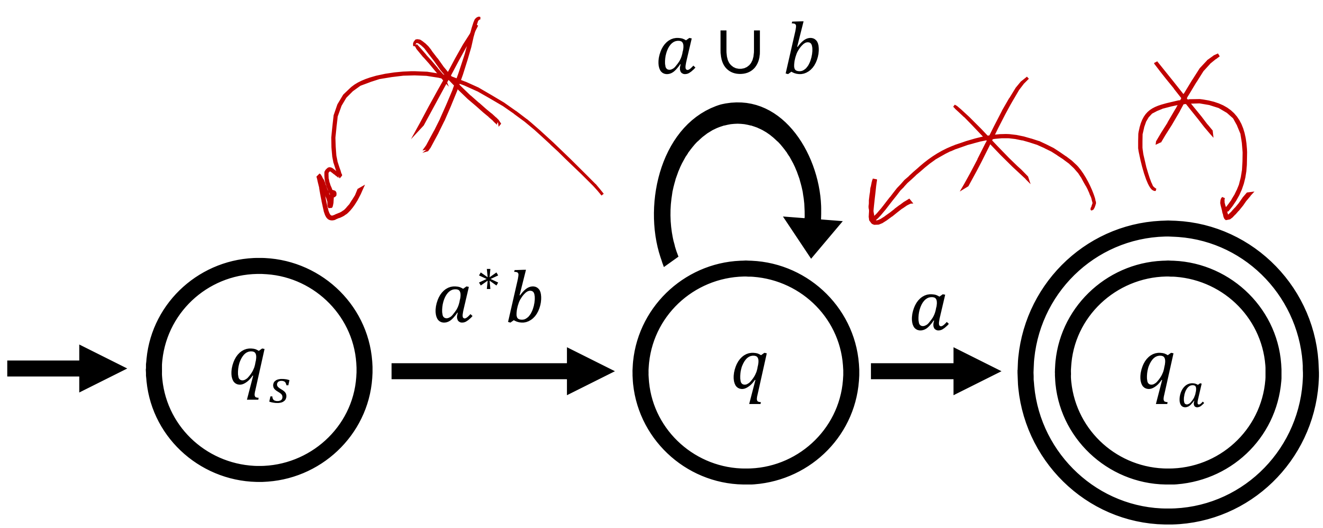
Theorem 2: Every NFA has an equivalent regex

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

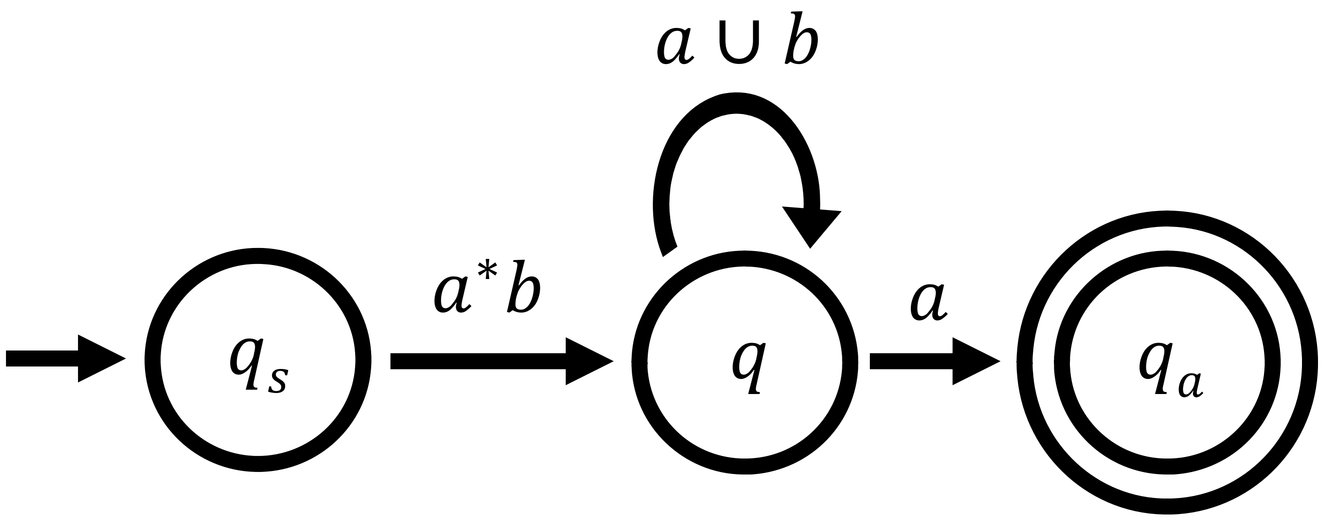


Generalized NFAs (GNFA)

- **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



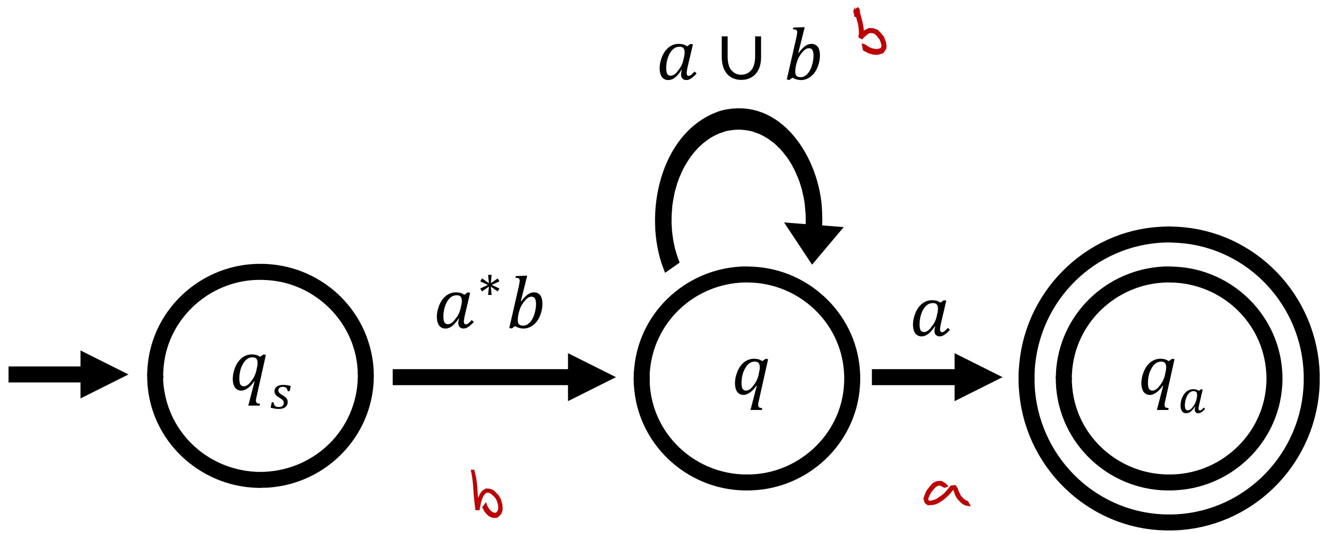
$$R(q_s, q) = a^* b$$

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

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

Which of these strings is accepted?

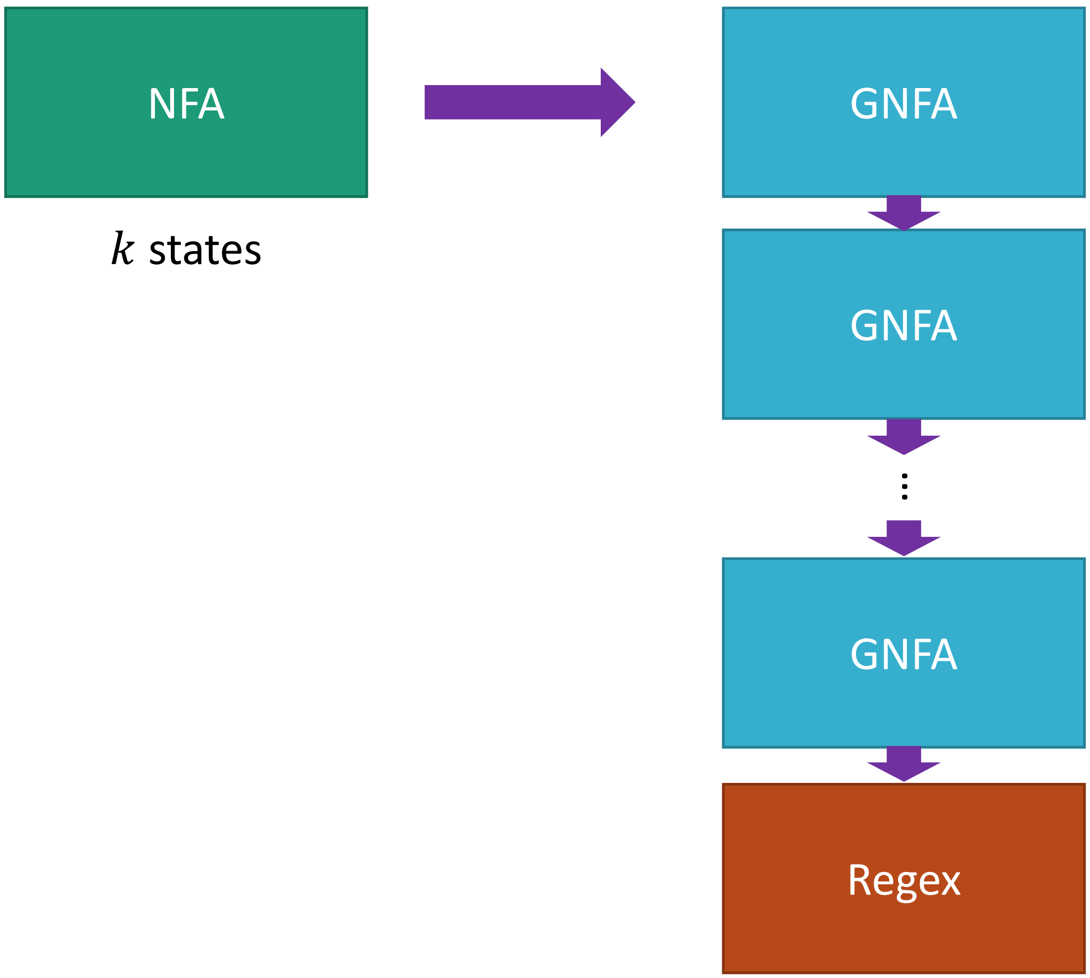
Which of the following strings is accepted by this GNFA?



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

NFA -> Regular expression

total # $= \sum_{i=3}^{k+2} i^2 \approx O(k^3)$



$k + 2$ states

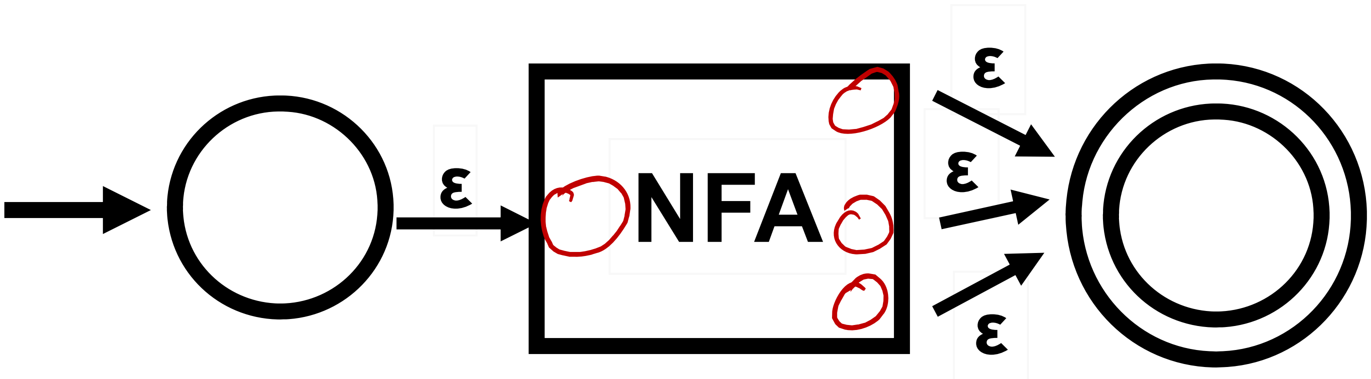
(perform $(k+2)^2$ replacements)

$k + 1$ states

$(k+1)^2$ replacements

2 states

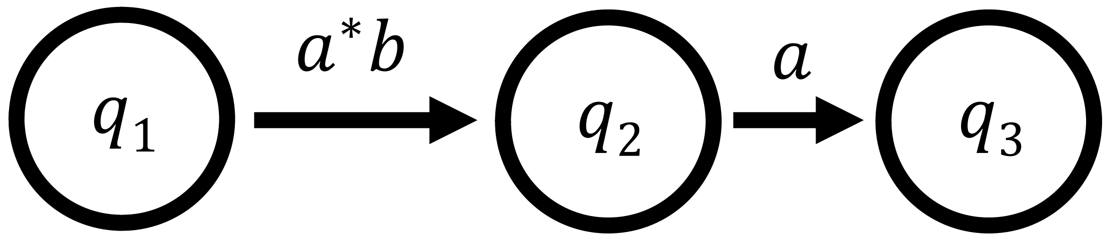
NFA -> GNFA

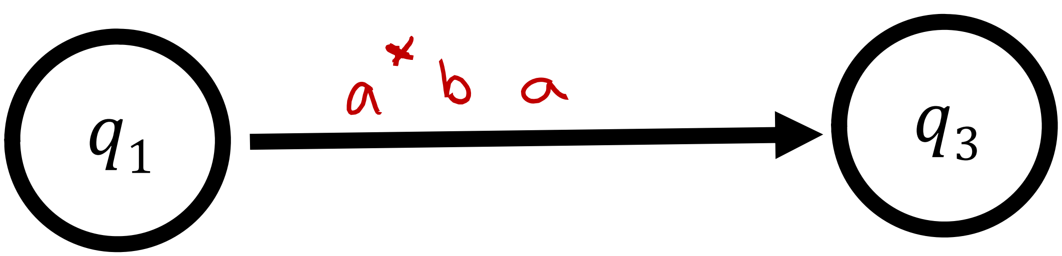


- Add a new start state with no incoming arrows.
- Make a unique accept state with no outgoing arrows.

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

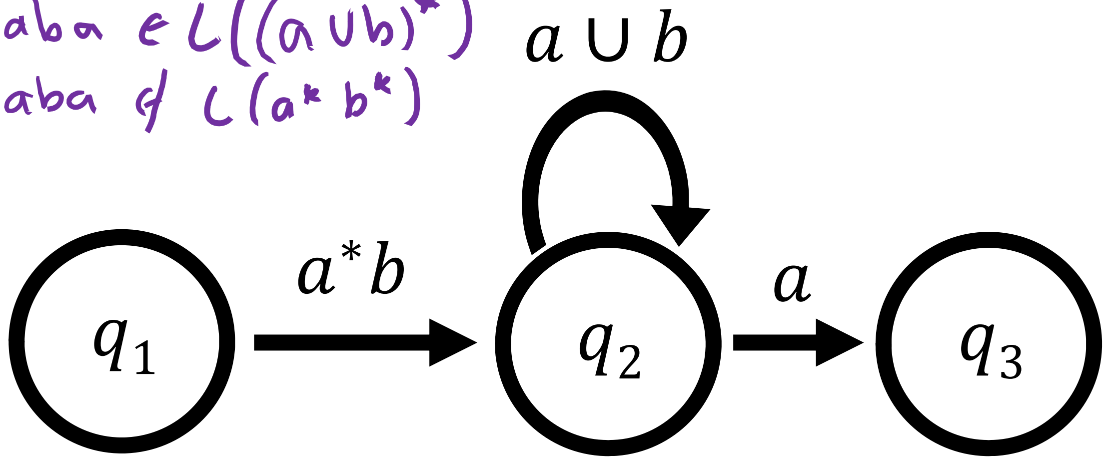
$\underbrace{a^*}\, b\, a^*\, b^*\, \underbrace{a}$

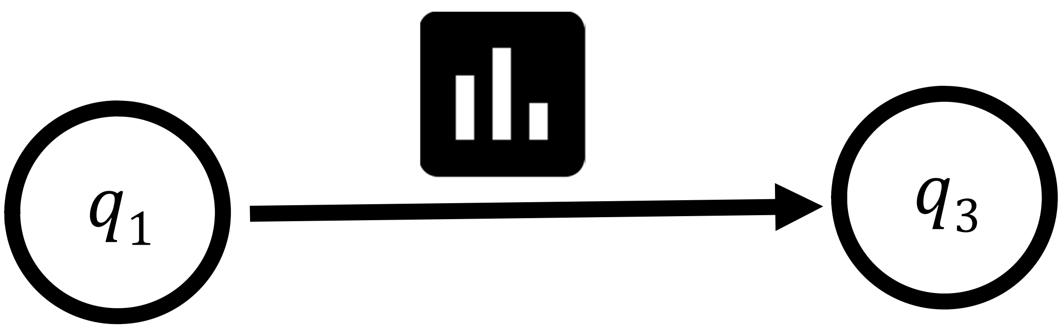
Idea: While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state

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

$aba \notin L(a^* b^*)$

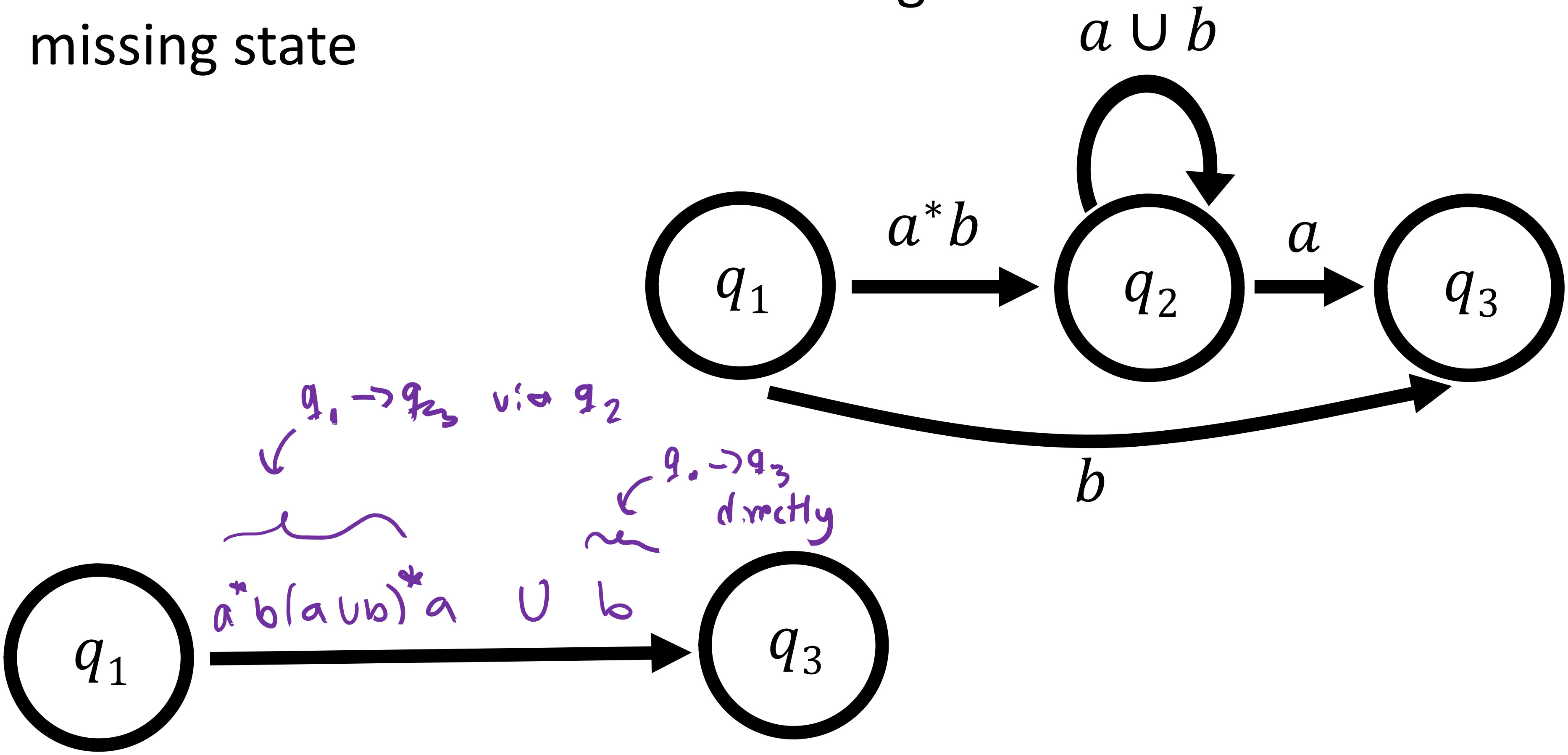
1) $a^*b(a \cup b)a$
2) $\underline{a^*b}(\underline{a \cup b})^*\underline{a}$
3) $a^*b \cup (a \cup b) \cup a$
4) None





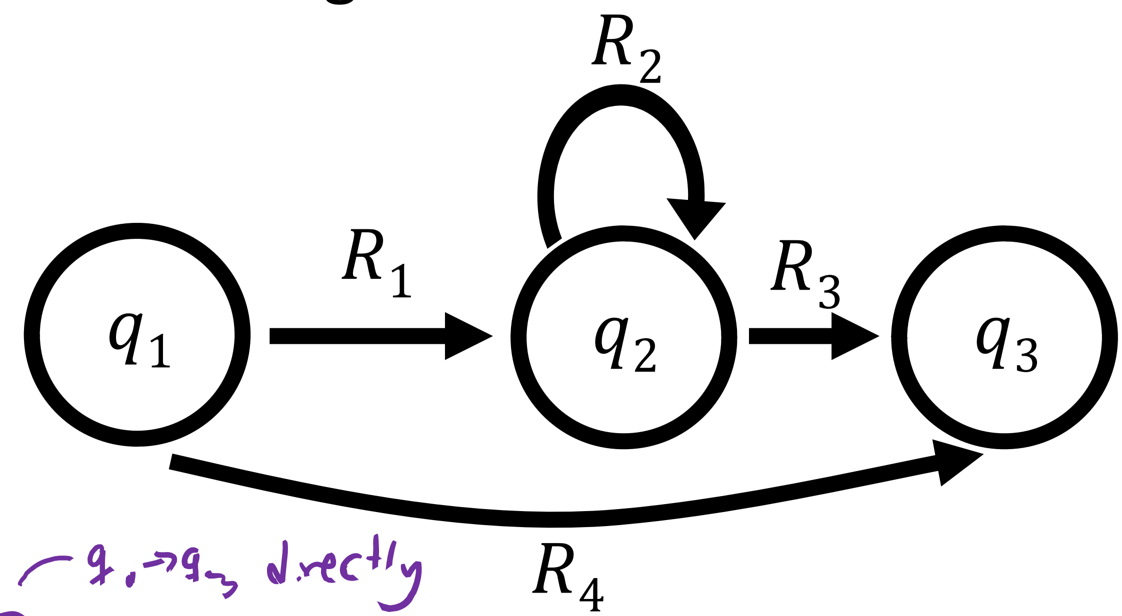
GNFA -> Regular expression

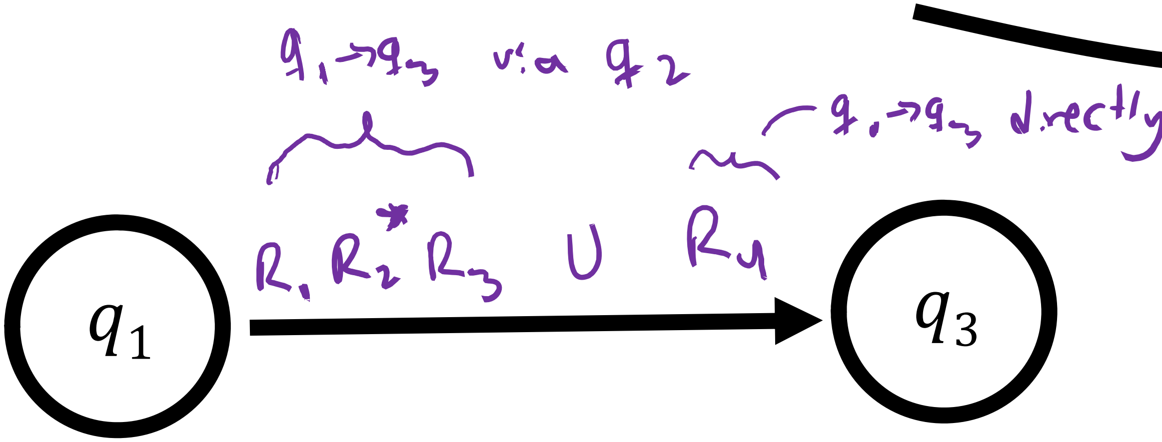
Idea: While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state

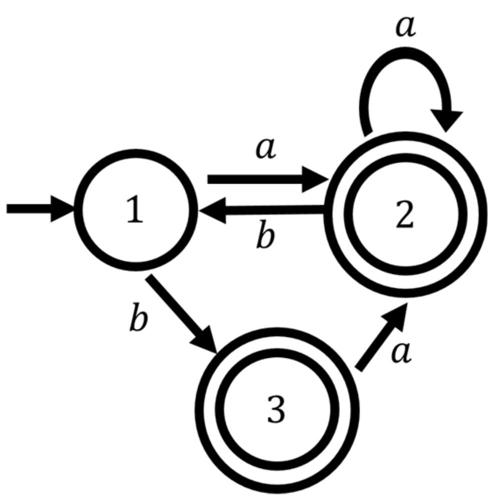


GNFA -> Regular expression

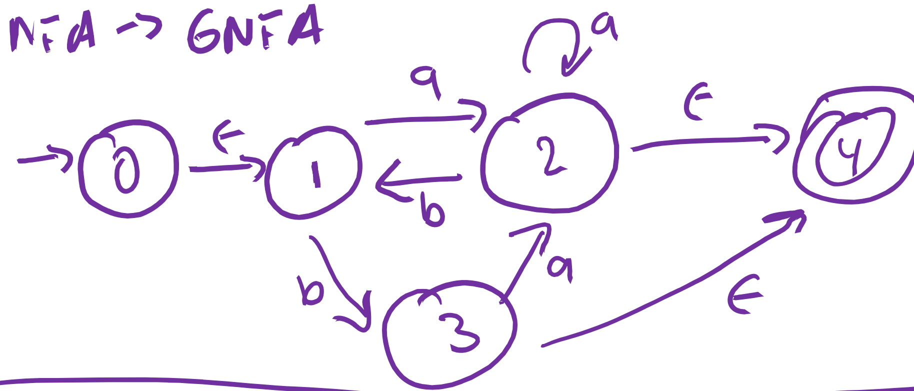
Idea: While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state



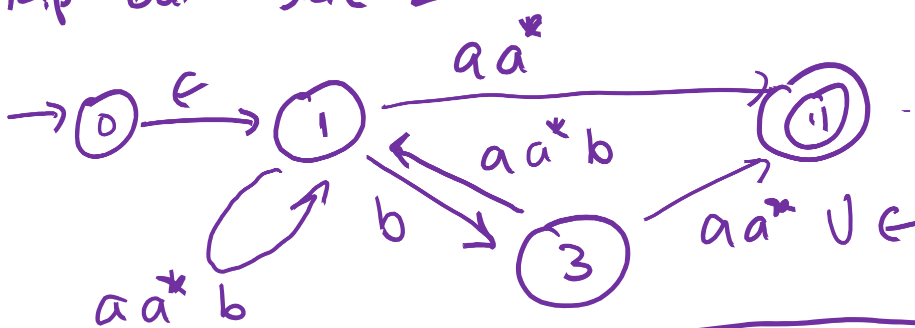




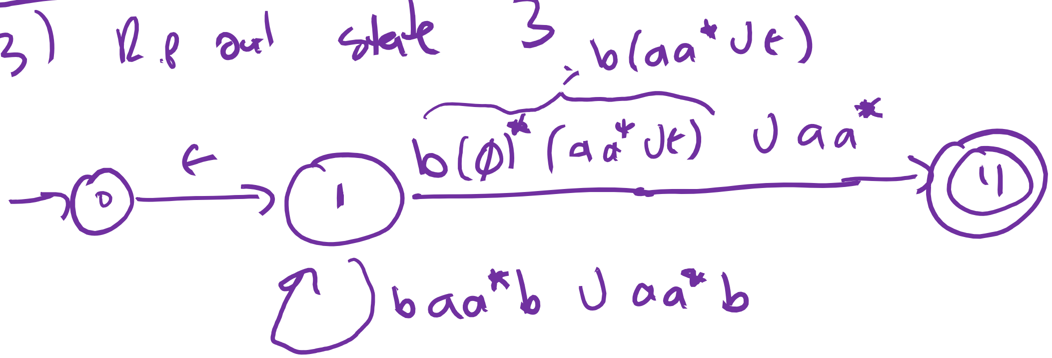
1) NFA → GNFA



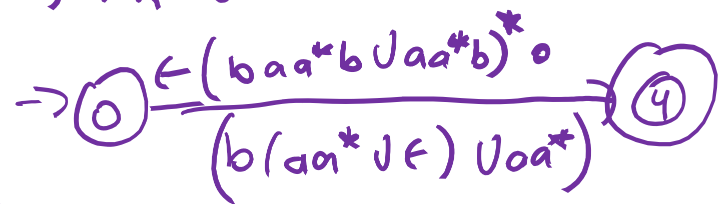
2) Rip out state 2



3) Rip out state 3



4) Rip out 1



Final regex

$(baa^*b \cup aa^*b)^*(b(aa^* \cup \epsilon) \cup aa^*)$

Non-Regular Languages

Motivating Questions

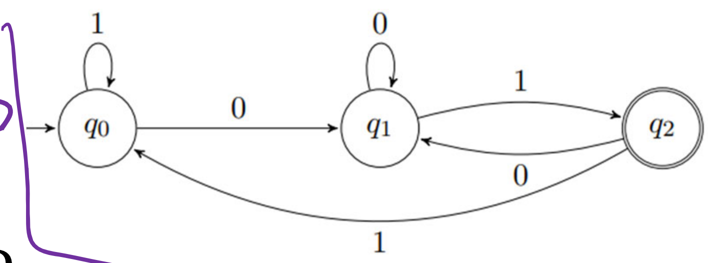
- We've seen techniques for showing that languages are regular

  DFA, NFA, regex

- 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

M → not necessarily



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

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

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

p_x : state M ends up in on ϵ

p_y : " on 0

p_z = " on 01

Claim: p_x, p_y, p_z all different

both accept & reject ✗

1) $p_z \neq p_x$, $p_z \neq p_y$ p_z accept, p_x, p_y are rejects

2) $p_x \neq p_y$ | Assume $p_x = p_y$ (for contradiction)

what happens when I run M on 1, 01

$p_x = p_y \xrightarrow{1}$

A General Technique

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

Definition: Strings x and y are **distinguishable** by L if there exists z such that exactly one of xz or yz is in L.

$\in \Sigma^*$

z is a dist. extension for x & y

Ex. $x = \varepsilon, \quad y = 0$

z = 1 is a dist. ext.

because $xz = \epsilon 1 = 1 \notin L$

$yz = 01 \in L$

Definition: A set of strings S is **pairwise distinguishable** by L if every pair of distinct strings $x, y \in S$ is distinguishable by L.

Ex. $S = \{\varepsilon, 0, 01\}$

$\epsilon, 0$ dist. (via $z = 1$)

$\epsilon, 01$ dist. (via $z = \epsilon$)

$0, 01$ dist. (via $z = \epsilon$)

A General Technique

Theorem: If S is pairwise distinguishable by L, then every DFA recognizing L needs at least $|S|$ states

Proof: Let M be a DFA with $< |S|$ states. By the pigeonhole principle, there are $x, y \in S$ such that M ends up in $\underbrace{\text{same state}}_{q}$ on x and y

Pigeons: x's in S
Holes: DFA states (q)
Pigeon x goes into hole q if M on input x ends up in state q

$\exists z \quad xz \in L, \quad yz \notin L$

M on x: $\xrightarrow{x} q \xrightarrow{z} p$

M on y: $\xrightarrow{y} q$

M on xz same state as M on yz

$\Rightarrow$ M doesn't recognize L

Back to Our Example

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

Theorem: If S is pairwise distinguishable by L, then every DFA recognizing L needs at least $|S|$ states

$S = \{\varepsilon, 0, 01\}$

$\left.\begin{array}{ll} \varepsilon, 0 & \text{dist} \\ 0, 01 & \text{dist} \\ \varepsilon, 01 & \text{dist} \end{array}\right\} \Rightarrow S$ pairwise dist.

Then $\Rightarrow$ Any DFA for A needs $\geq |S| = 3$ states

How to choose S?

1) w ends in 01 (except)
2) w ends in 0
3) Neither

$S' = \{1, 0, 01\}$ also works