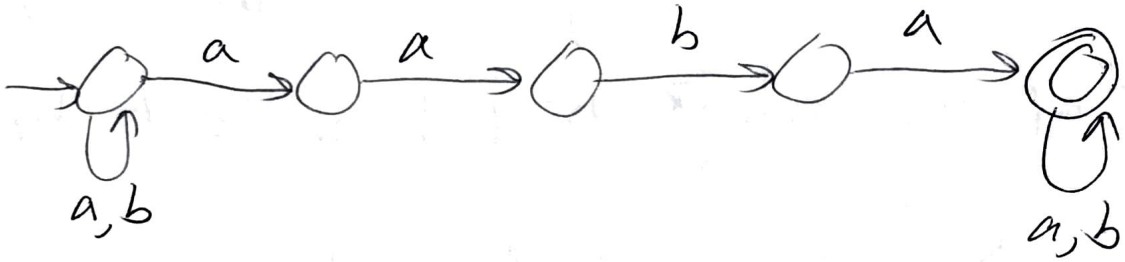
① Example of state elim method

CSCE 355
2/14/2022

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

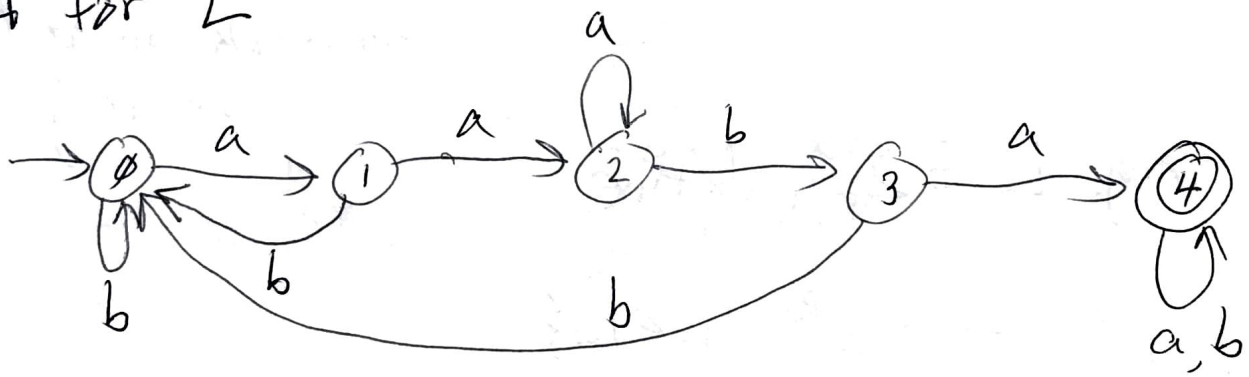
$L = \{w : w$ does not contain the string "aaba" as a substring$\}$

NFA for $\overline{L}$:



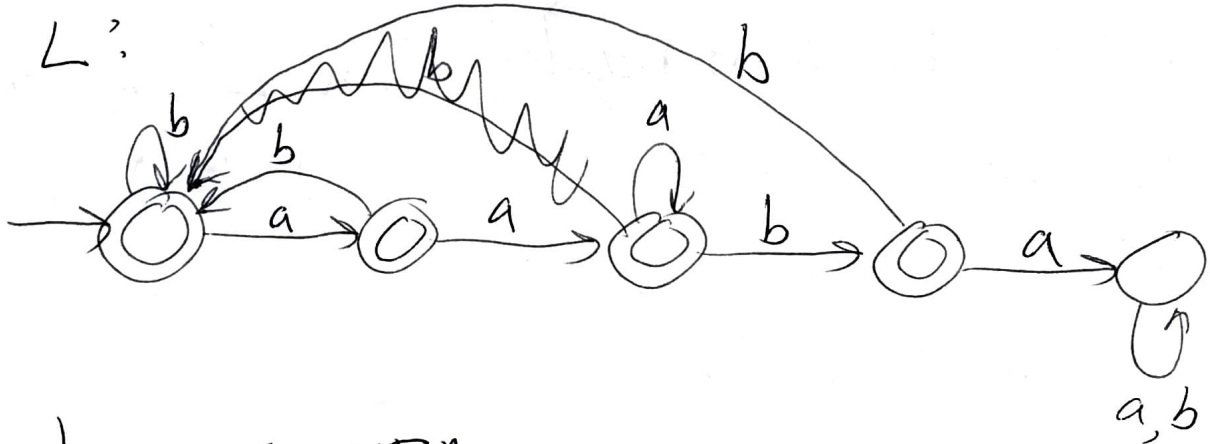
DFA for $\overline{L}$

A:

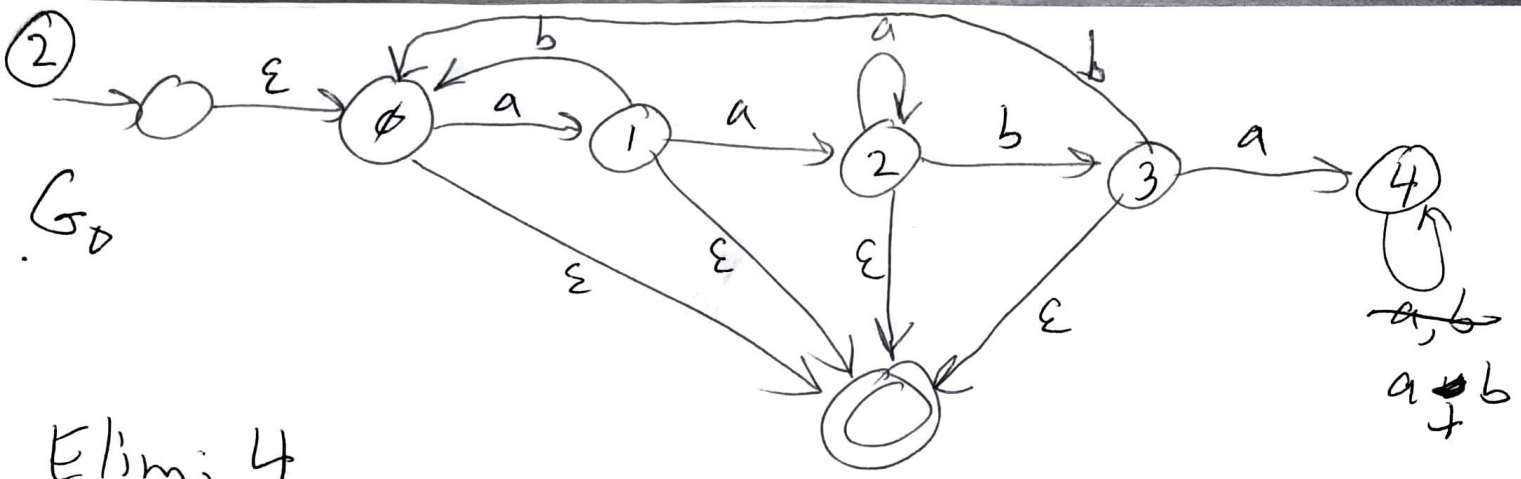


DFA L:

$\neg A$:

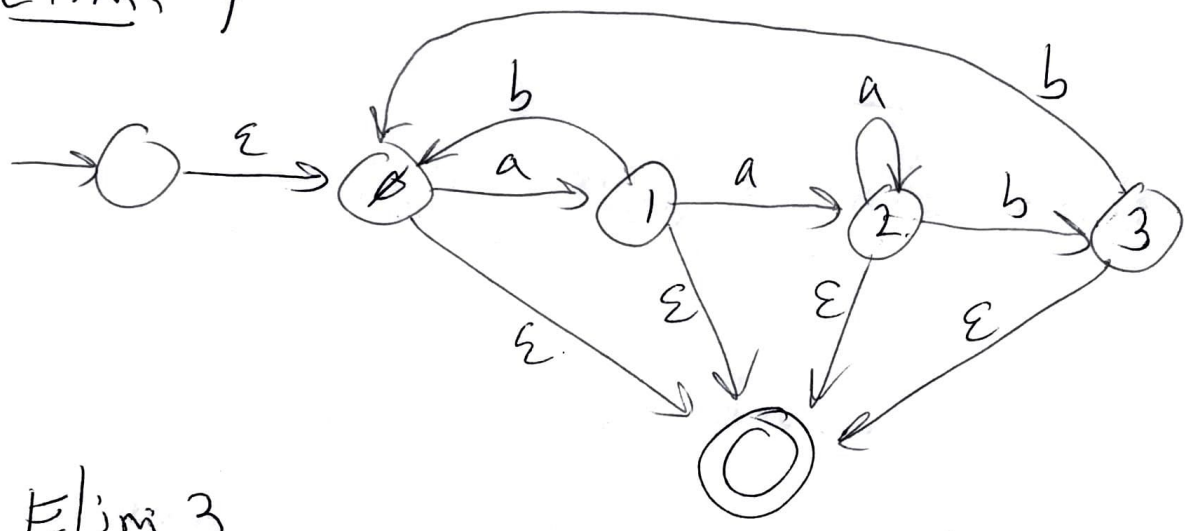


Equiv. clean ε-NFA:

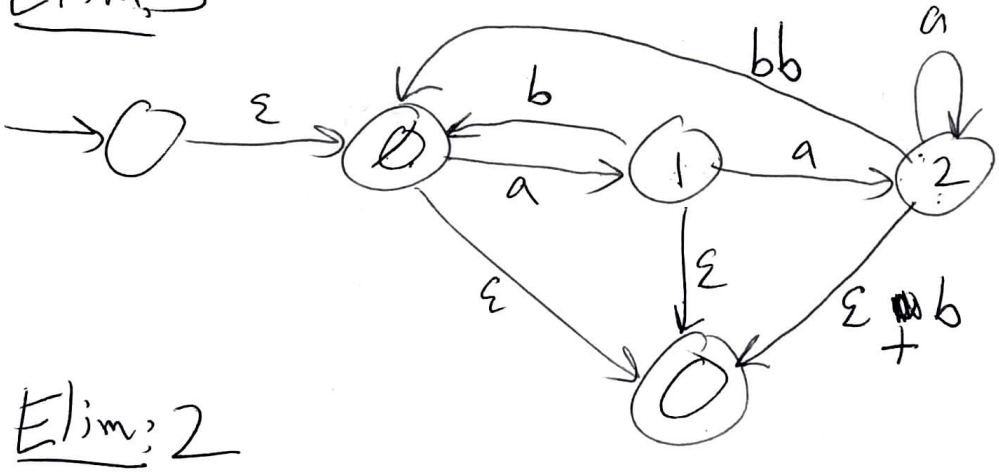


G_0

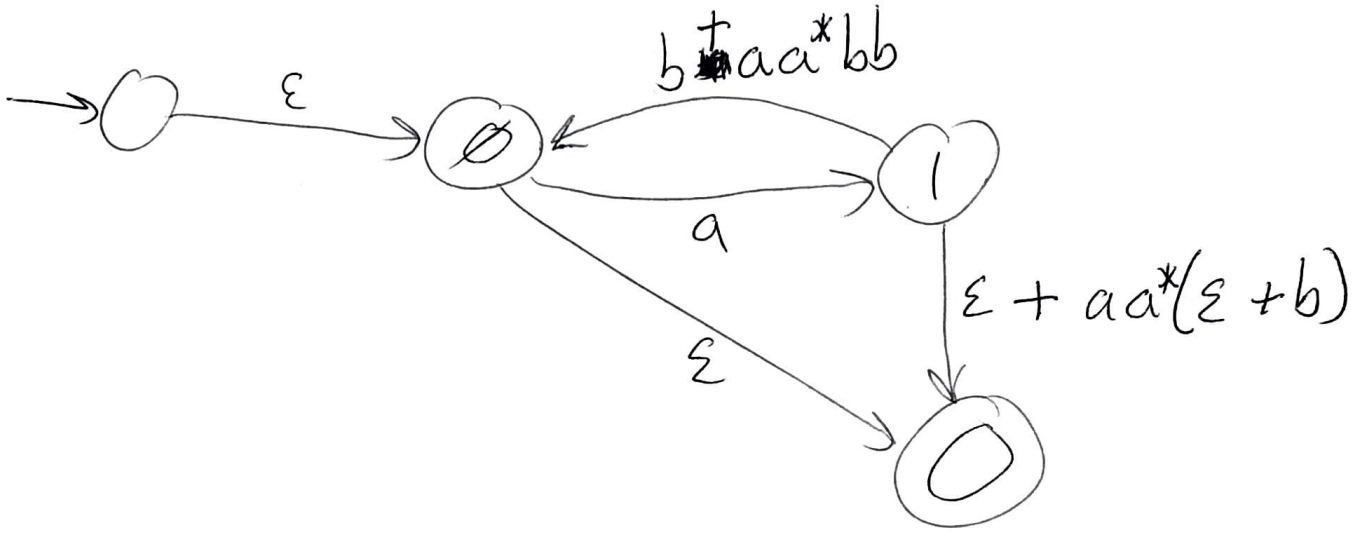
Elim; 4



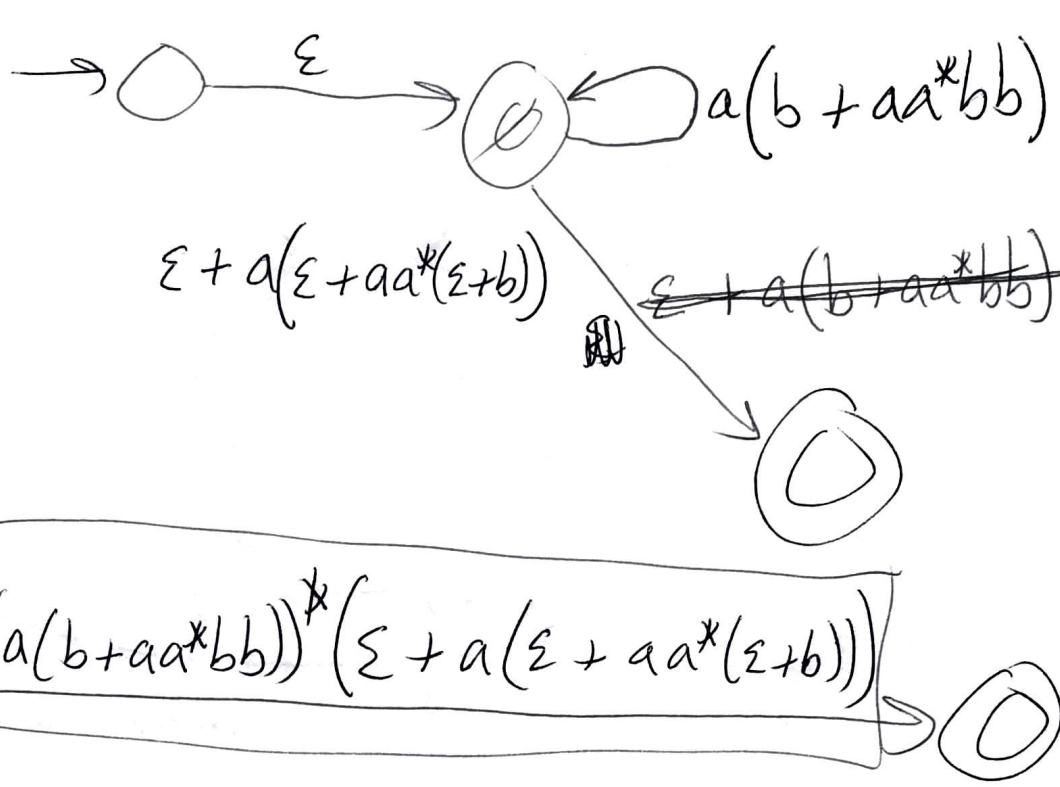
Elim; 3



Elim; 2



③ Elim: 1



Elim: ∅:

Conclusion: All these "devices" define the same class of languages (i.e., the regular langs.):

- DFAs
- NFAs
- ε-NFAs
- GNFAs
- regexes

Closure properties of the class of regular langs.

Ex: Reg langs are closed under intersection and complement.] ⟹ closed under union

(4)

$$L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$$

pf

$$\left.\begin{array}{l} L_1 = L(r) \\ L_2 = L(s) \end{array}\right\} L_1 \cup L_2 = L(r+s)$$

Def: Let w be a string (over some alphabet Σ)
Say $w = w_1 \cdots w_n$ where $n = |w|$ and each $w_i \in \Sigma$
We define

$$w^R := w_n \cdots w_1$$ (reversal of w; symbols in reverse order)

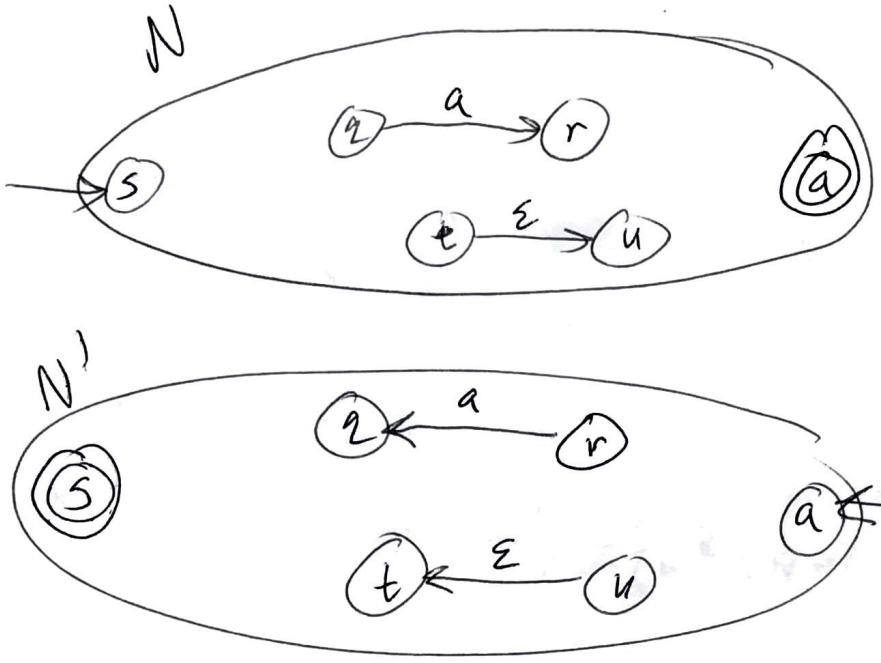
For lang. $L \subseteq \Sigma^*$, define

$$L^R := \{w^R : w \in L\}$$

Proposition: If L is regular, then L^R is regular.

Proof: Suppose L is regular. Let N be an ε-NFA recognizing L. WLOG, N has a unique accepting state (can make N clean, for example).

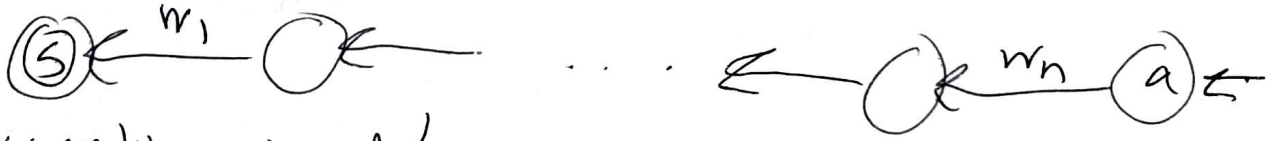
(5)



Then N' recognizes L^R

N accepts w $\iff$ There is an accepting path in N

$\iff$ There is



accepting in N'

$\iff$ N' accepts w^R ☐

Alternate proof: [Prev proof: clean ε-NFA for L $\Longrightarrow$ ε-NFA for L^R]

[Now: regex for L $\Longrightarrow$ regex for L^R]

Idea: Must see how the $()^R$ operator interacts

(6) with $\cup$, $+$, concat, and $*$:

L_1, L_2 langs

$$(L_1 \cup L_2)^R = L_1^R \cup L_2^R$$

$$\{w^R : w \in L_1 \text{ or } w \in L_2\} = \{w^R : w \in L_1\} \cup \{w^R : w \in L_2\}$$

$$(L_1 L_2)^R = L_2^R L_1^R$$

$$(L_1^*)^R = (L_1^R)^*$$

$x, y \in \Sigma^*$

$(xy)^R = y^R x^R$

Given any regex r

we construct a regex r'

such that $L(r') = L(r)^R$ by

the following recursive rules:

	r	r'
	$\emptyset$	$\emptyset$
$(a \in \Sigma)$	a	a
$(s, t$ regexes$)$	$s + t$	$s' + t'$
	st	$t's'$
	s^*	$(s')^*$

because $[(L_1 \cup L_2)^R = L_1^R \cup L_2^R]$

///

(7) $\left((a+bc)^*\right)' = \left((a+bc)'\right)^*$

$= (a' + (bc)')^*$

$= (a + c'b')^*$

$= (a + cb)^*$

Ex: $\left((a+bc)b^*a\right)' = ab^*(a+cb)$

Def: Let Σ and Γ be alphabets.

A ~~hom~~ string homomorphism from Σ to Γ is a map

$$\varphi : \Sigma^* \longrightarrow \Gamma^*$$

that preserves concatenation, i.e., $\forall x, y \in \Sigma^*$,

$$\varphi(xy) = \underbrace{\varphi(x)\varphi(y)}_{\text{concat}} \in \Gamma^*.$$

Note: if φ is a string homo., then $\varphi(\varepsilon) = \varepsilon$.

$$\underbrace{\varphi(\varepsilon)}_{\text{length 0}} = \varphi(\varepsilon\varepsilon) = \underbrace{\varphi(\varepsilon)}\,\underbrace{\varphi(\varepsilon)}$$

φ is completely determined by what it maps strings of length 1 to.