CSCE 355
2/12/2025

ε-NFA ⟶ regex (via GNFAs)

State elimination method

Given GNFA G_0: // with one accept state

$i = 1$; while G_{i-1} has an intermediate state:

Choose an intermediate state q of G_{i-1}
not the start state
" " accept "

Bypass q

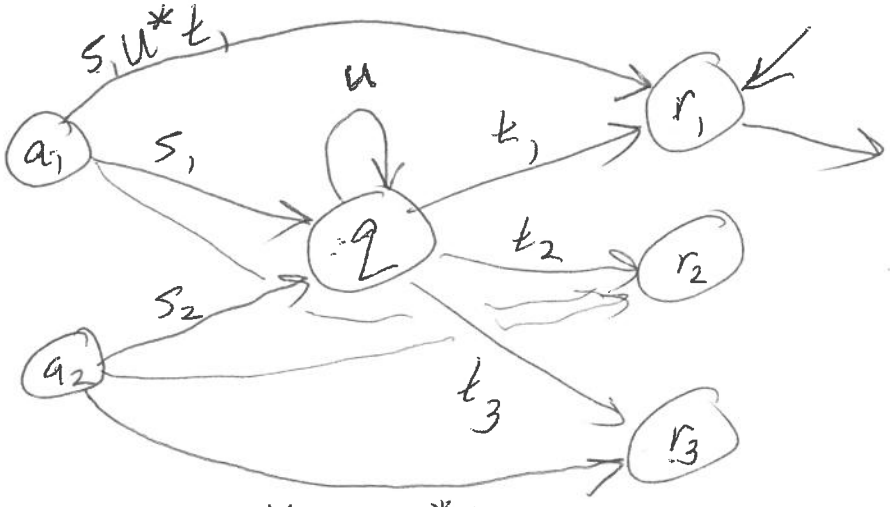
Remove q

Result is G_i // G_i is equiv to G_{i-1}

i++

end-while

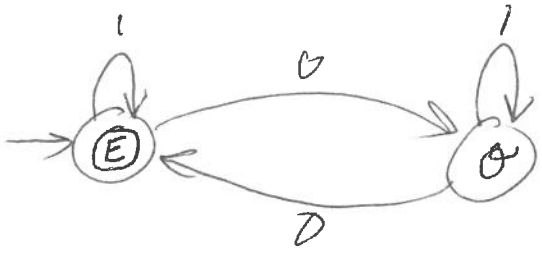
Bypassing a state s_i, t_j, u are regexes



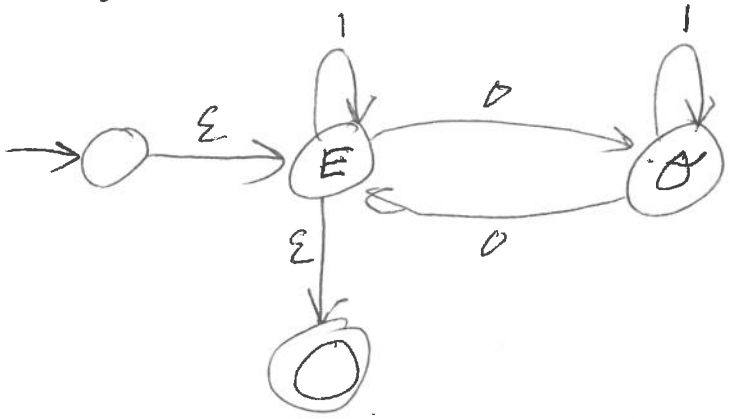
(* v was ~~label~~ label before the bypass)

Ex: $\Sigma = \{0, 1\}^*$

$L = \{w \in \Sigma : w \text{ has an even \# of zeros}\}$

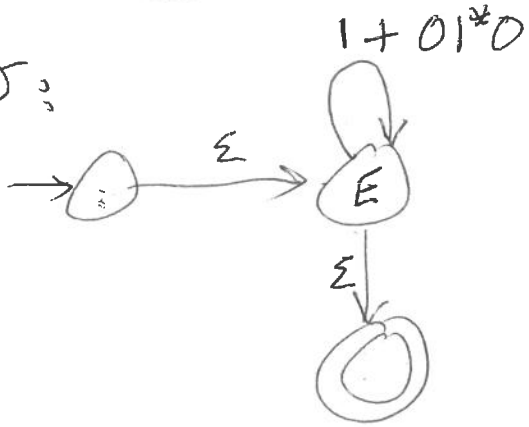


Make it clean:

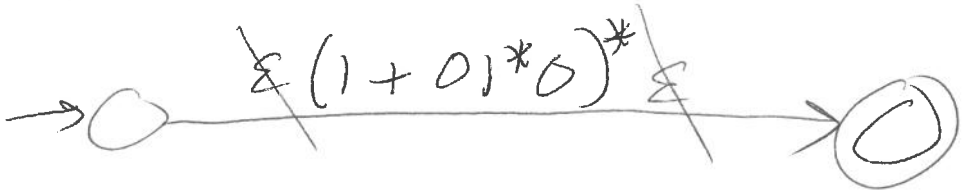


G_0

Remove O:



~~Remove~~ Remove E:



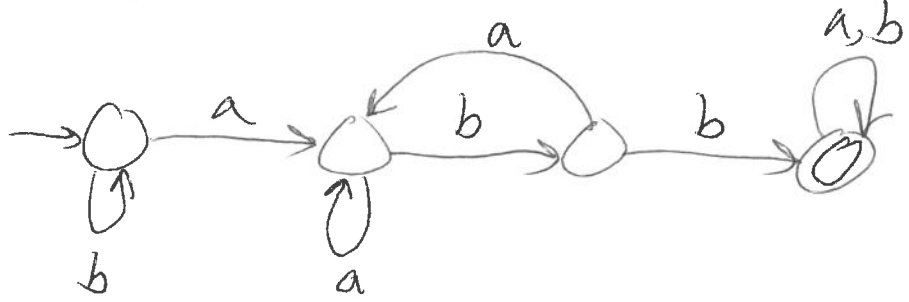
Equiv regex is $\boxed{(1 + 01^*0)^*}$

Ex: $L = \{w \in \{a,b\} : w$ does not have abb as a substring$\}$

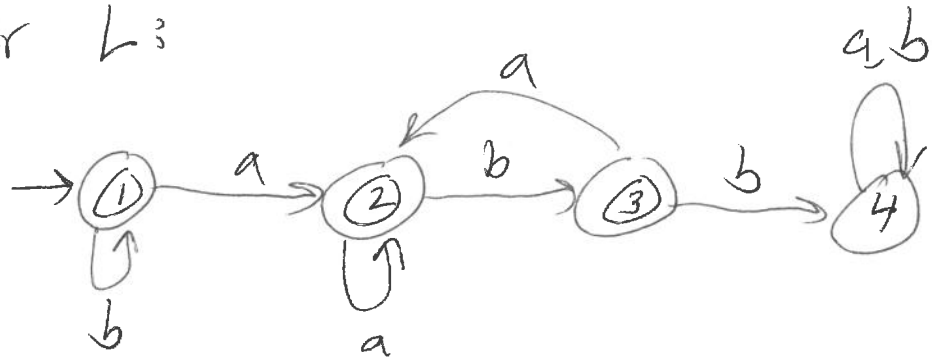
Regex for $\overline{L}$ is $(a+b)^*abb(a+b)^*$

Regex for L?

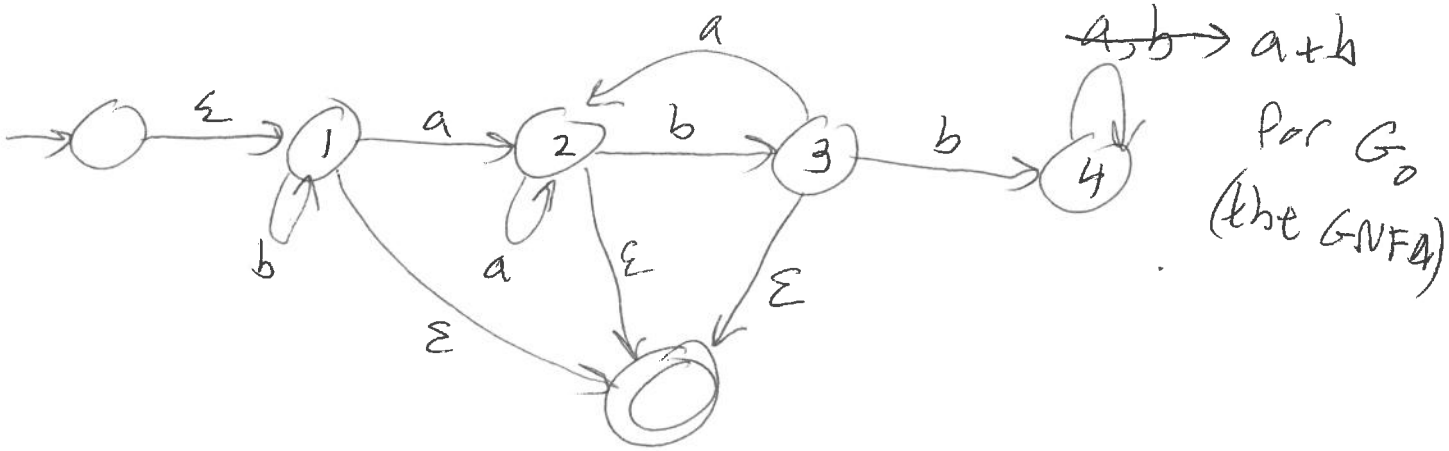
DFA for $\overline{L}$:



DFA for L:



Make an equivalent clean ε-NFA:



for G_0 (the GNFA)

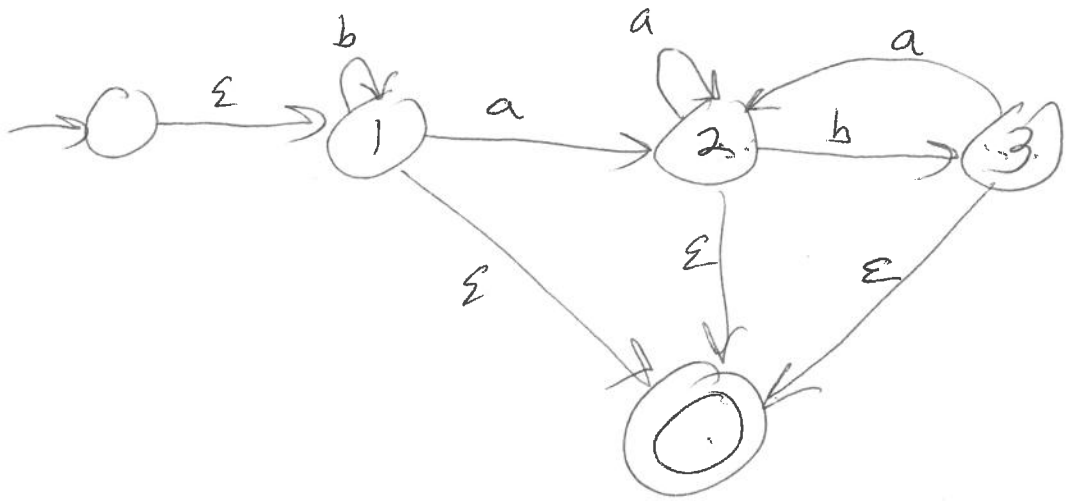
Start eliminating states.

Strategy to minimize workload & length of the final answer: remove states that requires the fewest # bypasses first.

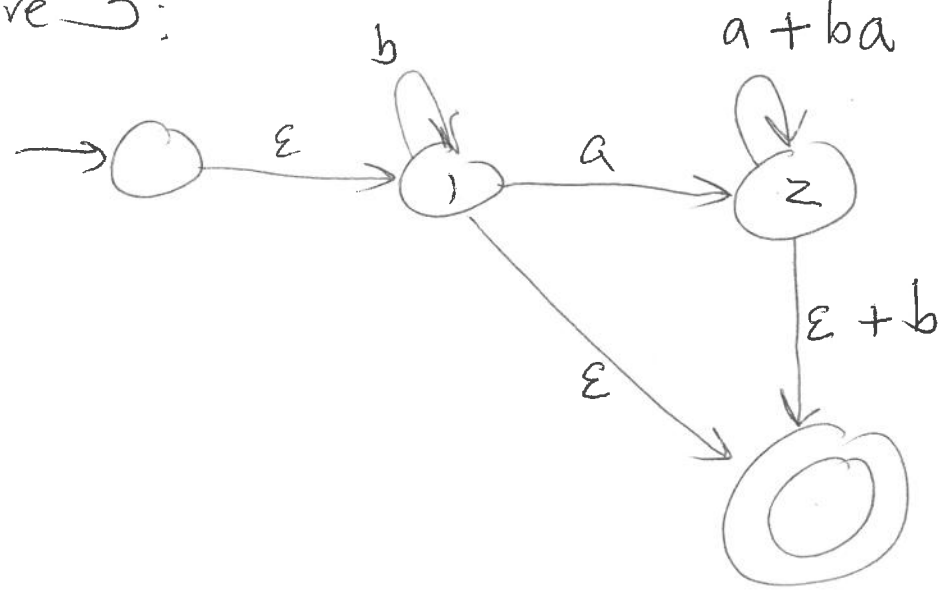
bypasses = (# incoming edges) × (# outgoing edges)

not incl. any self-loop

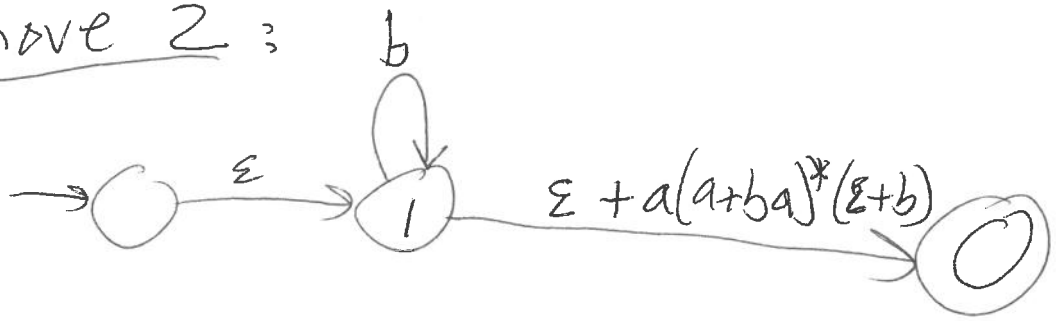
Remove 4 (no bypasses needed):



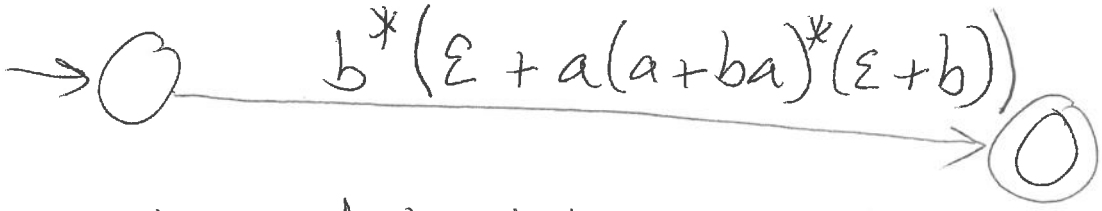
Remove 3:



Remove 2:



Remove 1:



No more intermediate states —
return the regex on the → ○ → ◎ edge

Answer: $b^*(\varepsilon + a(a+ba)^*(\varepsilon+b))$

Thm: TFAE (the following are equivalent)
for any language $L \subseteq \Sigma^*$:

L is regular:

1. $L = L(A)$ for some DFA A ← sets-of-states construction
2. $L = L(N)$ for some NFA N ← ε-move removal
3. $L = L(M)$ for some ε-NFA M ← trivial
4. $L = L(G)$ for some GNFA G ← state-elimination
5. $L = L(r)$ for some regex r.

We've essentially proved this by construction

(1) → (3) is trivial
(5) → (3) last Monday

Closure properties of the reg. langs.

~~Lang~~ Language ops that preserve regularity.

We already know some of these;

1. complementation (if L is regular, then $\overline{L}$ is regular)

2. ~~Boolean~~ Intersection (if L_1, L_2 are reg. then so is $L_1 \cap L_2$)

3. (1,2) $\Rightarrow$ all Boolean set ops: $\cup$, $\setminus$, Δ

   $\setminus$ ↑ relative complement

4. Concatenation: If langs L_1, L_2

$$L_1 L_2 = \{ wx : w \in L_1 \ \& \ x \in L_2 \}$$

Prop: If L_1, L_2 are reg. then so is $L_1 L_2$

Proof: Let r_1, r_2 be regexes such that $L(r_1) = L_1$ and $L(r_2) = L_2$. Then

$r_1 r_2$ is a regex, and $L(r_1 r_2) = L(r_1) L(r_2) = L_1 L_2$.

$\therefore L_1 L_2$ is regular, given by regex $r_1 r_2$. //

5. $*$-operator: L a language.

Define $L^* := \{\varepsilon\} \cup L \cup LL \cup LLL \cup \cdots \cup L^n \cup \cdots$

Prop: If L is regular, then L^* is regular.

Proof: If r is a regex for L, then $r*$ is a regex for L^*. //

---

string reversal: For string $w = w_1 \cdots w_n$ ($w_i \in \Sigma$ for all $i \in \{1, \ldots, n\}$), then define the reversal of w as

$$w^R := w_n w_{n-1} \cdots w_1$$

For language $L \subseteq \Sigma^*$, define

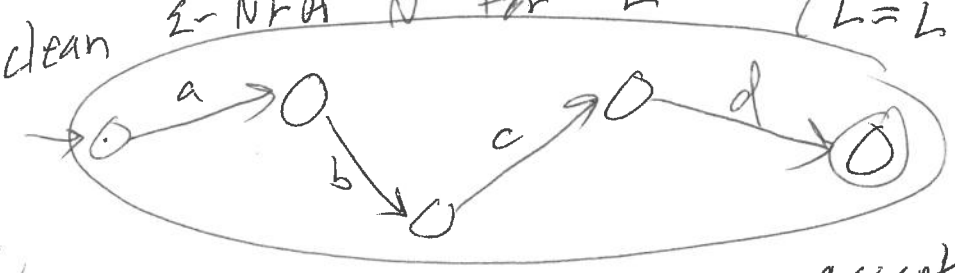
$$L^R := \{w^R : w \in L\}$$

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

Proofs: (2 proofs)

Proof 1: Convert a clean ε-NFA for L into a (clean) ε-NFA for L^R.

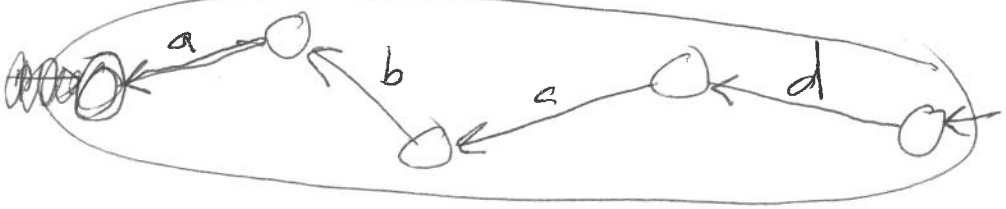
clean ε-NFA N for L ($L = L(N)$)



accepts $w = abcd$

swap start & accepting states, reverse all arrows

ε-NFA for L^R:



Proof 2': Convert an arbitrary regex r to a regex r' such that $L(r') = L(r)^R$, by induction on the syntax of r:

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

s, t are regexes ←