CSCE 355
2/1/2023

Regular Expressions

Fix any alphabet Σ.

A <u>regular expression</u> (<u>regex</u> for short) over Σ is built from primitive exprs using 3 operators.

Primitive (atomic) regexs	Language denoted by a regex
$\emptyset$	$\emptyset$
a (any $a \in \Sigma$)	$\{a\}$

operators: $+$, concat, $*$

Let s, t be regexes

$\rightarrow s + t$	union of the langs of s & t
concat $\rightarrow st$	concatenation of the langs
Kleene $*$-operator, Kleene closure $\rightarrow s^*$	s concat with itself any number of times (zero or more)

$ab + c$	denotes	$\{ab, c\}$
$ab + c^*$	"	$\{ab, \varepsilon, c, cc, ccc, \ldots\}$
$(a+b)(a+b)$	"	$\{aa, ab, ba, bb\}$

<u>Def:</u> Fix alphabet Σ. Let L_1, L_2 be any languages over Σ. The concatenation L_1L_2 of L_1 and L_2 is the language

$$L_1L_2 = \{xy : x \in L_1 \text{ and } y \in L_2\}$$

Language concatenation is associative:

$$(L_1L_2)L_3 = L_1(L_2L_3)$$
$$= \{xyz : x \in L_1, y \in L_2, z \in L_3\}$$

For any string $x \in \Sigma^*$ and $n \in \mathbb{Z}$, $n \geq 0$,

$$x^n := \underbrace{x \cdots x}_{n \text{ times}} \qquad \text{want } x^{m+n} = x^m x^n$$

True if $m, n > 0$. What if m, n are zero?

define $x^0 := \varepsilon$ by convention.

Then $x^{m+n} = x^m x^n$ holds for all $m, n \geq 0$.

If L is a language, then $L^n = \underbrace{LL \cdots L}_{n \text{ times}}$

want $L^{m+n} = L^m L^n$ to hold for all $m, n \geq 0$.

Define $L^0 := \{\varepsilon\}$

<u>Def</u>: For Language $L \subseteq \Sigma^*$, define

$$L^* := L^0 \cup L^1 \cup LL \cup LLL \cup \cdots \cup L^n \cup \cdots$$

$L = \{ab\}$: $L^* = \{\varepsilon, ab, abab, ababab, \ldots\}$

$L = \{a, b\}$ $L^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$

[Σ^* is special case of the same rule: all strings over Σ]

<u>Def</u>: For regex r over Σ, let $L(r)$ be the language of r (lang denoted by r). $L(r)$ is defined inductively by the following rules:

	r	$L(r)$	
	$\emptyset$	$\emptyset$	atomic regexes
(any $a \in \Sigma$)	a	$\{a\}$	
Any regexes s, t,	$s+t$	$L(s) \cup L(t)$	
	st	$L(s)L(t)$	
	s^*	$L(s)^*$	

Ex: $L((ab+bb)^*) = \{\varepsilon, ab, bb, abab, abbb, bbab, bbbb, \dots\}$

$$L(a+b) = L(a) \cup L(b)$$
$$= \{a\} \cup \{b\} = \{a, b\}$$

similarly, $L(a+b+c+\dots) = \{a, b, c, \dots\}$

$$L(ab) = L(a)L(b) = \{a\}\{b\} = \{ab\}$$

similarly

$$L(abc\dots) = \{abc\dots\}$$

$$L(\emptyset) = \emptyset$$

$$L(\emptyset^*) = \emptyset^0 \cup \emptyset \cup \emptyset\emptyset \cup \dots$$
$$= \{\varepsilon\}$$

Use ε as a regex denoting the language $\{\varepsilon\}$

$(\varepsilon := \emptyset^*)$

Notes about precedence & associativity of operators:

$$\left.\begin{array}{ll} + & \text{(union)} \\ \cdot & \text{(concat)} \end{array}\right\} \text{associative}$$

$*$ unary postfix

Precedences (lowest to highest): $+$, concat, $*$

so: $ab^* = a(b^*)$

parens can be used to force grouping:

$$(ab)^*$$

$$ab^* = \{a, ab, abb, abbb, \ldots\}$$

$$(ab)^* = \{\varepsilon, ab, abab, ababab, \ldots\}$$

Common shorthands ("syntactic sugar")

Let r be a regex.

$$\text{""} := \varepsilon := \emptyset^*$$

$$r^+ := rr^* \quad (= r + rr + rrr + \ldots)$$

("one or more r's")

$$r? := r + \varepsilon \quad (= r + \emptyset^*)$$

("optional r", zero or one r's)

Let Σ be the ascii char set

~~for~~ $[abc] := a+b+c$ <u>character class</u>

$= [cba]$

$[A-Z] := A+B+C+\cdots+Z$ (subrange)

$[A-Za-z]$:= all upper & lowercase letters

More generally, "abc+def" = {abc+def}

"?"

Matching: A string w matches a regex r means $w \in L(r)$. (also, "r matches w")

More on character classes:

[^abc] complement: matches all single ascii chars except a, b, and c.

. (period) match any single char (except \n) newline

In linux apps, + only used as unary operator ("one or more")

For union, use | (vertigule) instead

Regex pattern that matches: (1) all legal identifiers in C/C++/Java:

[A-Za-z_][A-Za-z0-9_]*

(2) unsigned int constants:

[0-9]+ (= [0-9][0-9]*)

(3) Floating point constants in Pascal

[0-9]+"."[0-9]+([eE][+-]?[0-9]+)?

C allows 3. or .3 or 3e10

All expressed ~~in terms~~ in terms of $\emptyset$, $a \in \Sigma$, $+$, concat, $*$

Next: regex $\rightleftarrows$ ε-NFA

CSCE 355
2/6/2023

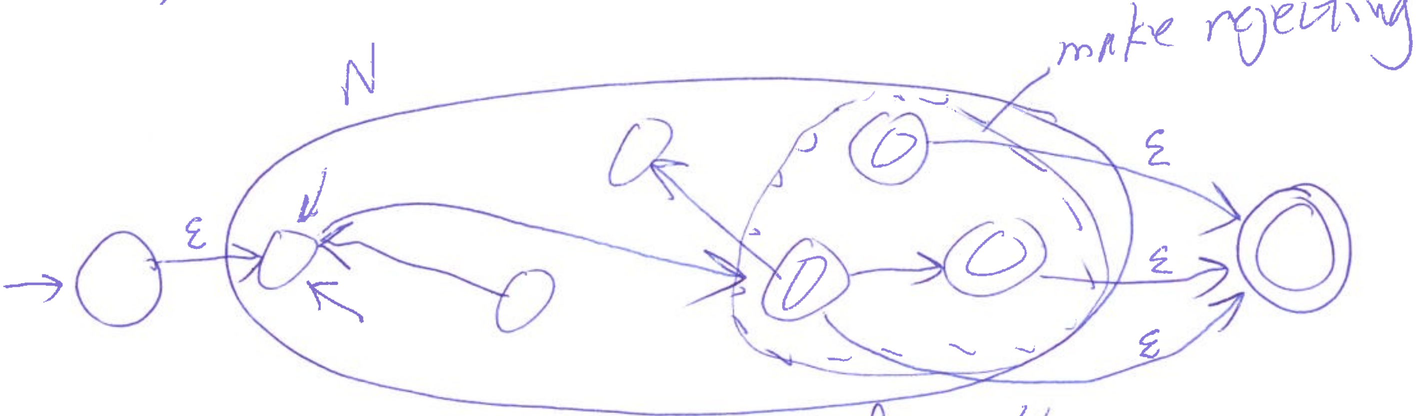
regex $\rightleftarrows$ ε-NFA

<u>Prop</u>: For every regex r (over fixed alphabet Σ), there exists an equivalent clean ε-NFA N
(i.e., $L(N) = L(r)$)

<u>Def</u>: An ε-NFA is <u>clean</u> if
1) it has exactly one accepting state, and it is not the start state.
2) there are no transitions into the start state
3) " " " " out of the accepting state.

<u>Lemma</u>: For every ε-NFA there is an equivalent clean NFA.

<u>Pf</u>: By construction. Given ε-NFA N:



Proof of correctness omitted. //

Proof of the Prop: By induction on the length of r.

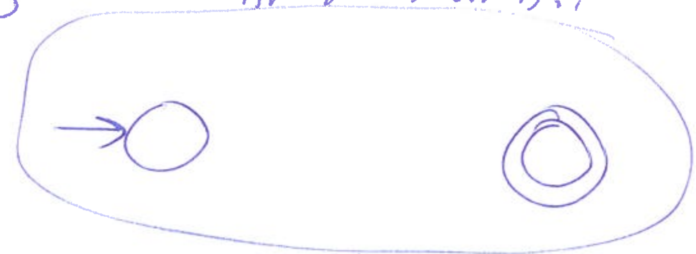
Base cases: r is atomic, i.e., $r = \emptyset$ or $r = a$ (some $a \in \Sigma$)

Recall: $L(\emptyset) = \emptyset$ (empty lang.)

$L(a) = \{a\}$

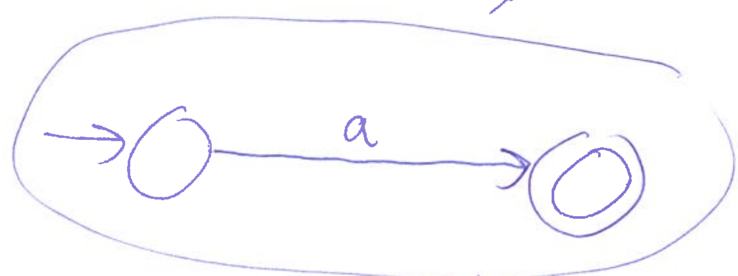
no transitions!!

If $r = \emptyset$ then $N :=$



$L(N) = \emptyset$

If $r = a$ then $N :=$



one transition

Inductive cases: r is not atomic, so

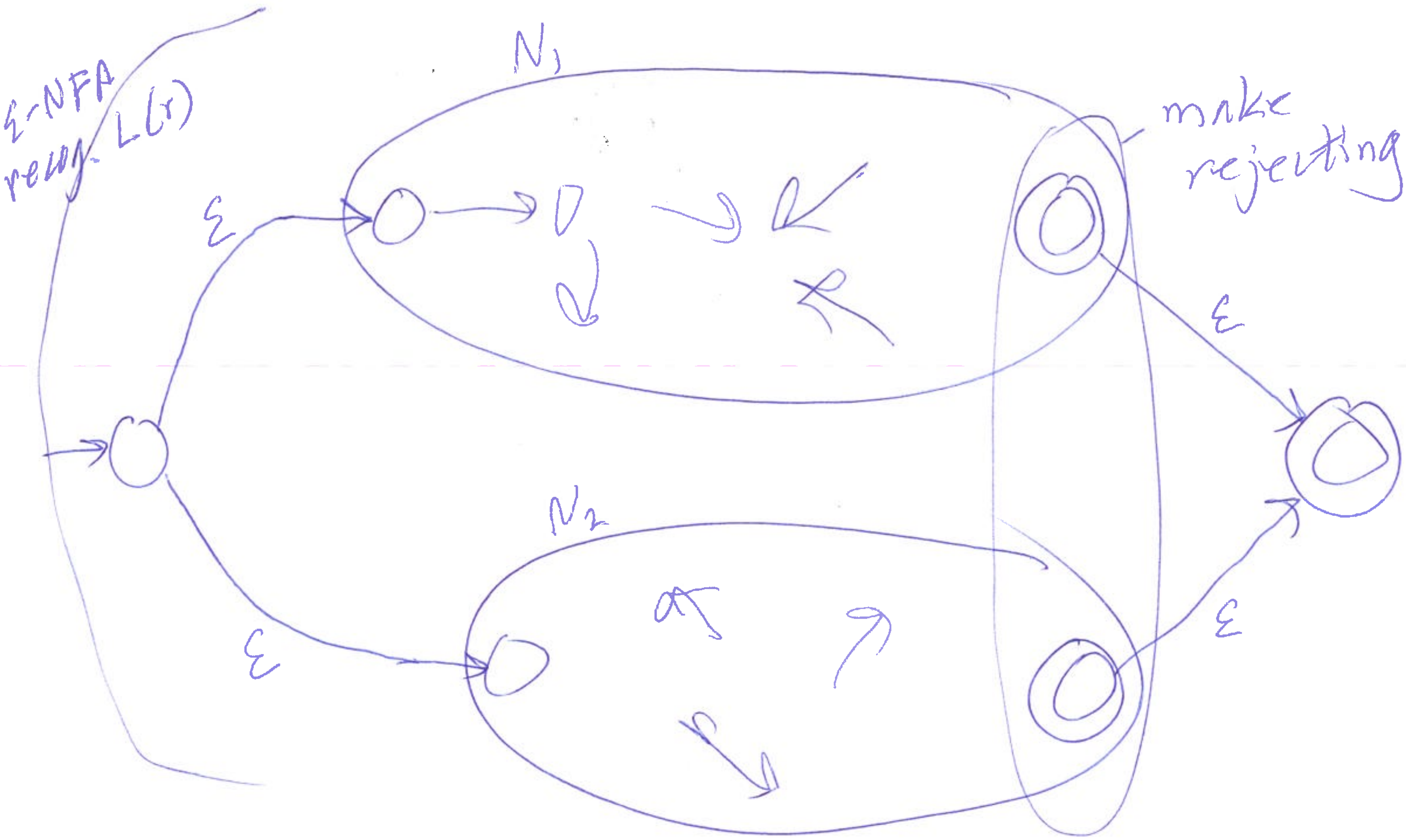
$r = s + t$ for some regexes s, t

$r = st$ " " " "

$r = s^*$ for some regex s

Case 1: $r = s + t$ so $L(r) = L(s) \cup L(t)$

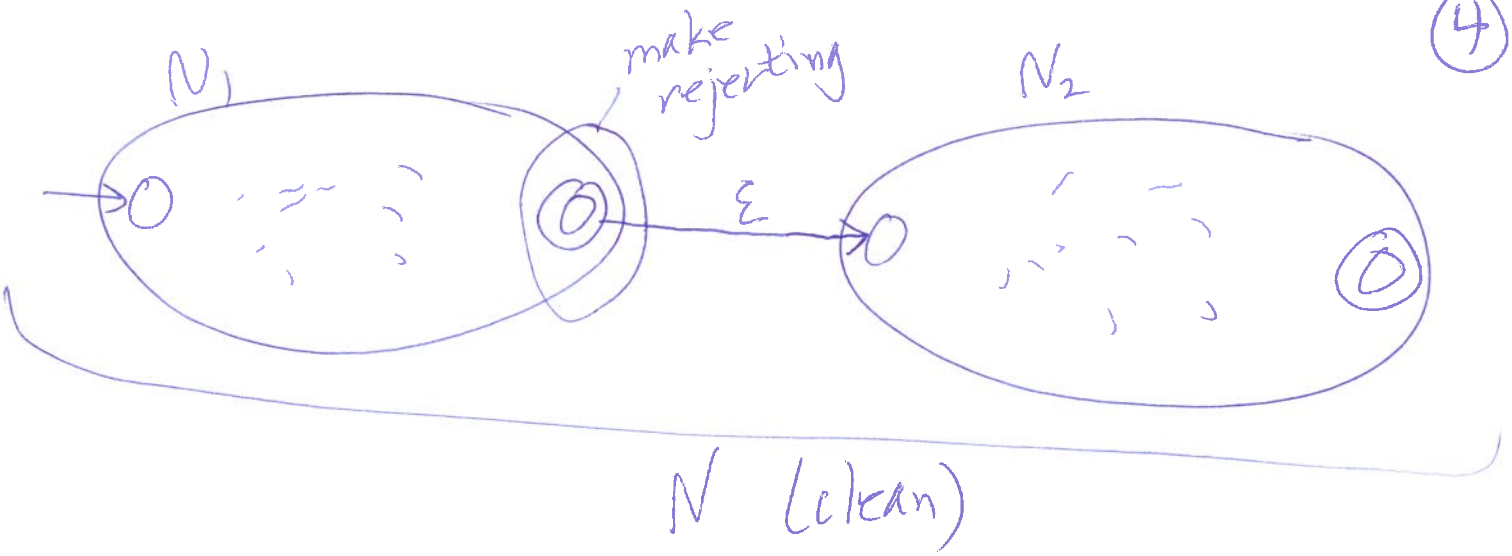
Ind hyp: there are clean ε-NFAs N$_1$ for s and N_2 for t $(L(s) = L(N_1)$ and $L(t) = L(N_2))$



Case 2: $r = st$ $L(r) = L(s)L(t)$

$= \{xy : x \in L(s) \text{ and } y \in L(t)\}$

Ind hyp: Let $L(N_1) = L(s)$ and $L(N_2) = L(t)$ for some ε-NFAs N_1 & N_2, clean. Build ε-NFA N s.t. $L(N) = L(r)$, as follows:

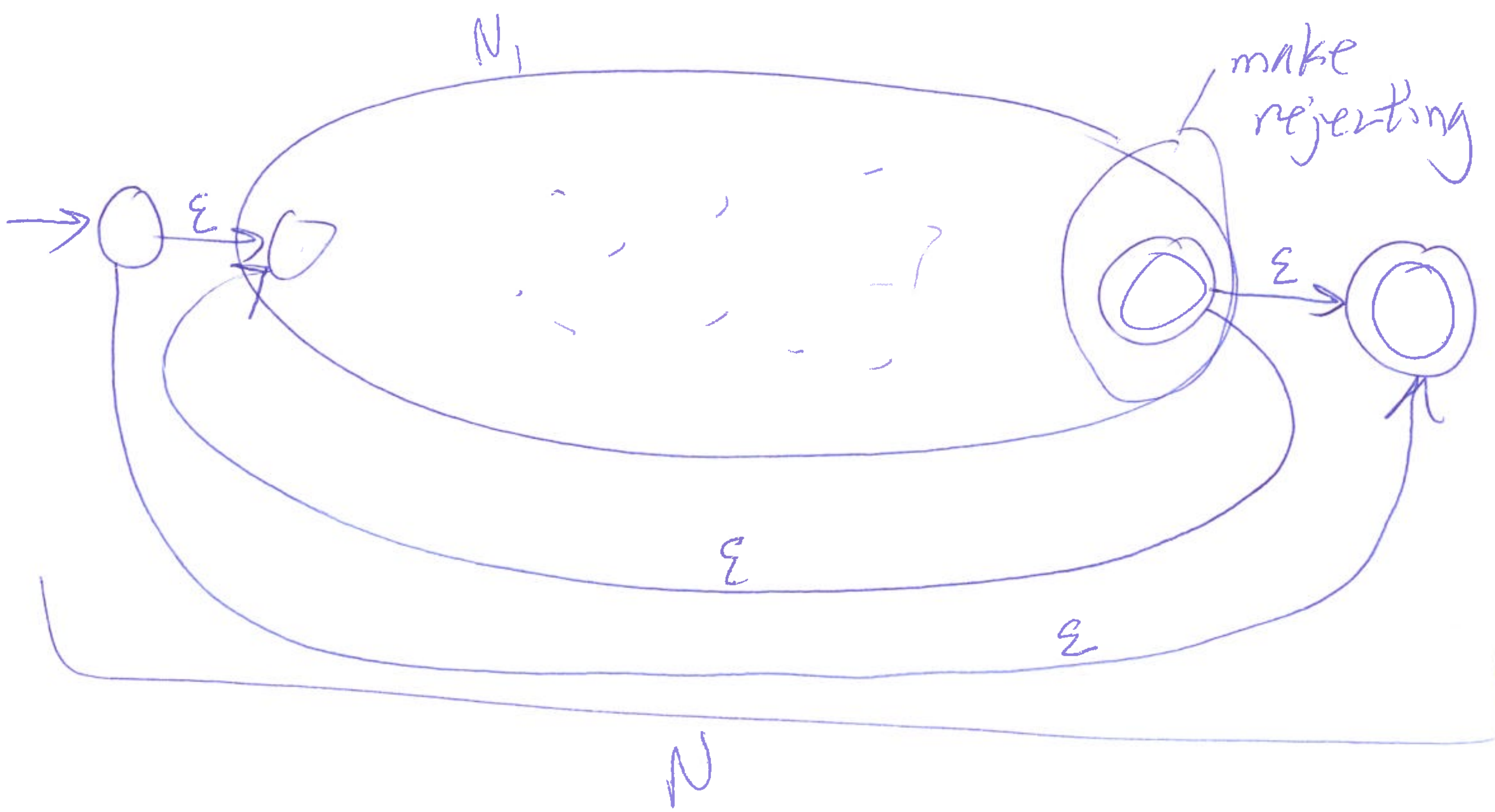


Case 3: $r = s^*$

$(L(r) = \{\varepsilon\} \cup L(s) \cup L(s)L(s) \cup L(s)L(s)L(s) \cup \cdots)$

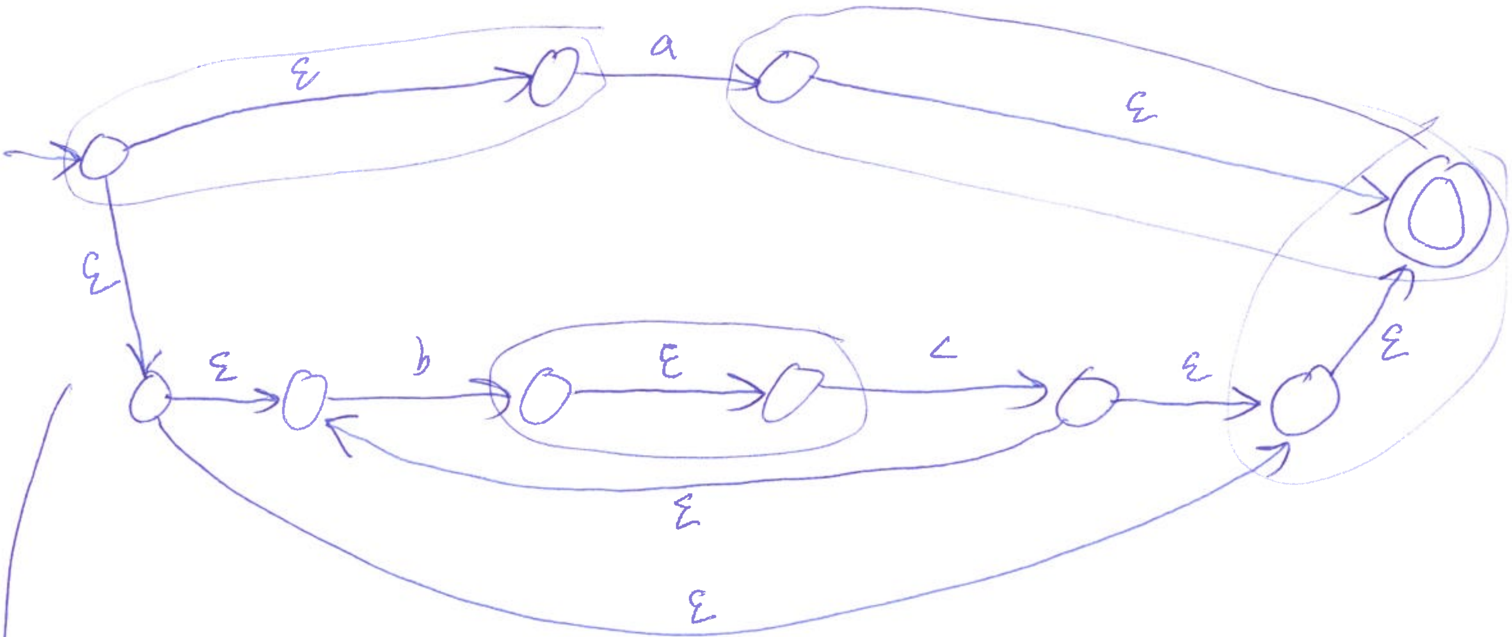
Ind hyp: Given a clean ε-NFA N_1 recog. $L(s)$.

Want to build a clean ε-NFA N recog. $L(r) = L(s^*)$.

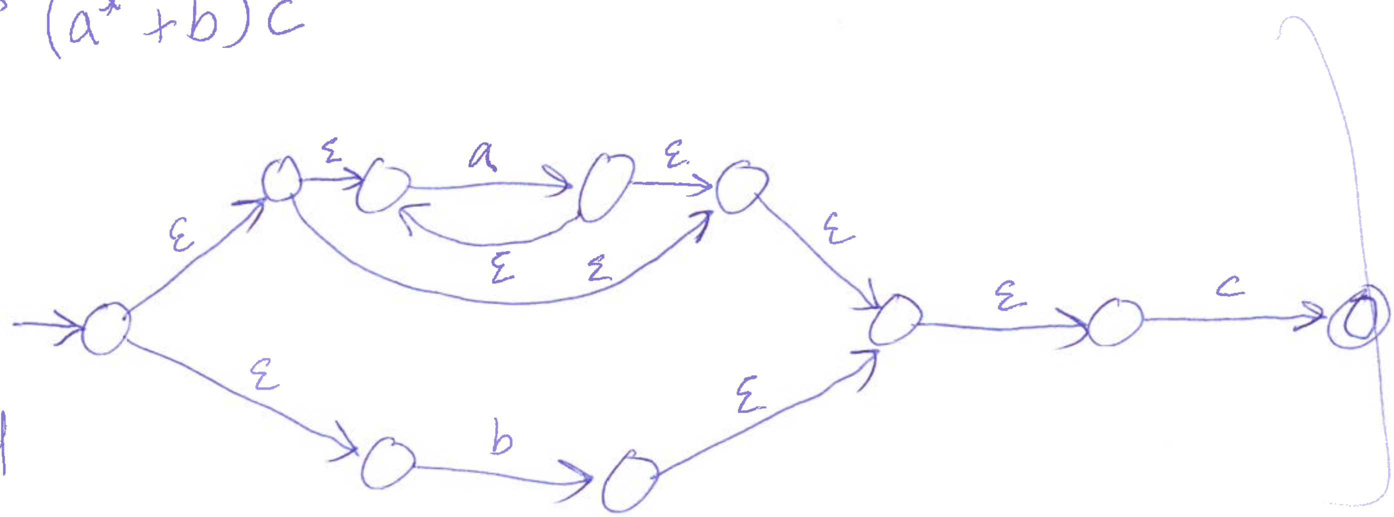


Proof of correctness (oral). ■

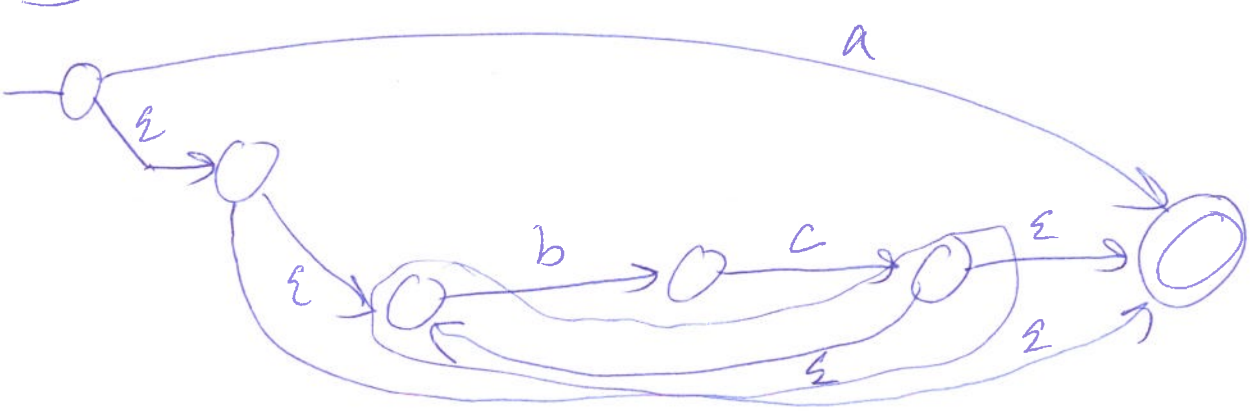
Ex: $a + (bc)^*$ **Hint:** work from the bottom up (atomic regexs $\xrightarrow{\text{combine}}$ . . . $\rightarrow$ final regex)



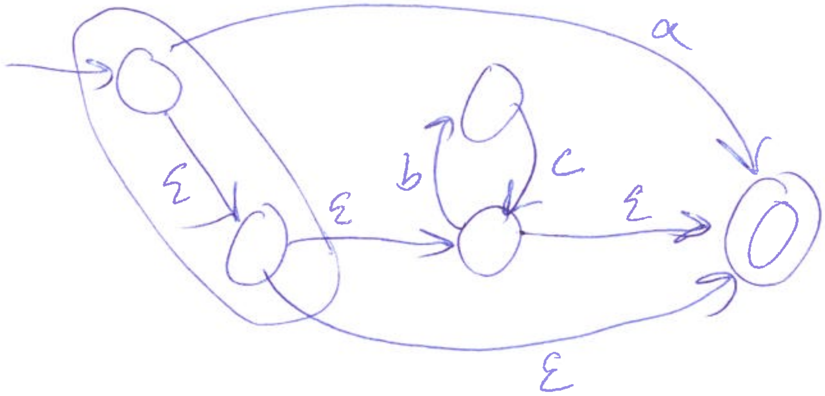
Ex: $(a^* + b)c$



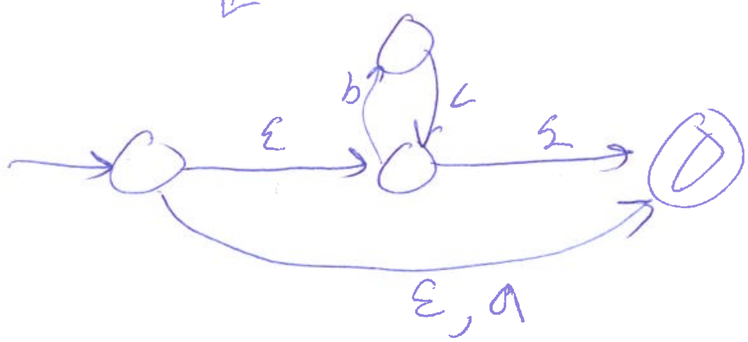
~~optimize~~ improved



$\Downarrow$



$\Downarrow$



can do:

regex $\Longrightarrow$ ε-NFA $\Longrightarrow$ NFA $\Longrightarrow$ DFA

- regex ⇒ ε-NFA: today
- ε-NFA ⇒ NFA: last week
- NFA ⇒ DFA: set-of-states

Cor: For any regex r, $L(r)$ is regular.

CSCE 355
2/8/2023

ε-NFA $\Rightarrow$ regex today

Start with some ε-NFA N

$$N \longrightarrow \underset{N}{\text{clean}} \longrightarrow G_0 \to G_1 \to G_2 \to \cdots \longrightarrow G_n \to \underset{r}{\text{regex}}$$

<u>State elimination method</u>

$(L(r) = L(N))$

A generalized NFA (GNFA) has transitions labeled by regexes.

$$\cdots \longrightarrow q \xrightarrow{ab^* + c} r \longrightarrow \cdots$$

Can get directly from q to r by reading some string from the input that matches ab^*+c, i.e., a followed by 0 or more b's or a single c.

~~None~~ <u>Note</u>: An ε-NFA is "essentially" a GNFA: ~~Jis~~ Just interpret the transition diagram as that of a GNFA.

$$q \xrightarrow{a} r$$

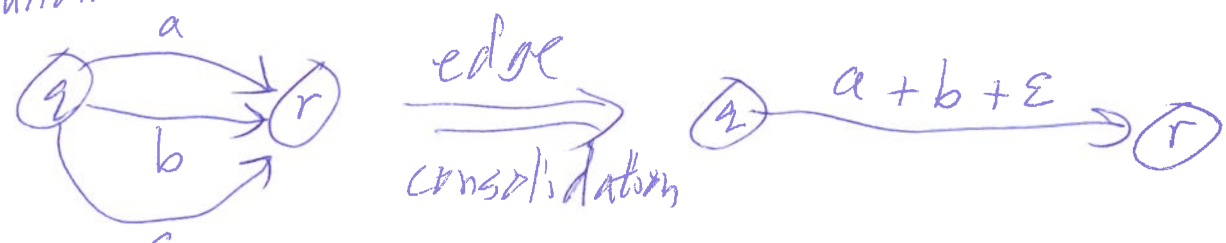
$\varepsilon := \emptyset^*$

$L(\varepsilon) = \{\varepsilon\}$

<u>except</u> a GNFA does not allow multiple edges from a given state to a given state;

<u>Ex:</u> ε-NFA: GNFA

allowed



not allowed in a GNFA

<u>Construction</u>: Given ε-NFA N

1. Make N clean
2. Let G_0 be the equiv. GNFA to N
3. For $i := 1$ to ...n.. (stop when G_{i-1} has no intermediate states, i.e., only the start state & accepting states remain)

   G_{i-1} is given

   - Pick some intermediate state q in G_{i-1}
   - Remove q from G_{i-1}
   - Add bypassing edges

   } state elimination

– Consolidate multiple edges if necessary ③

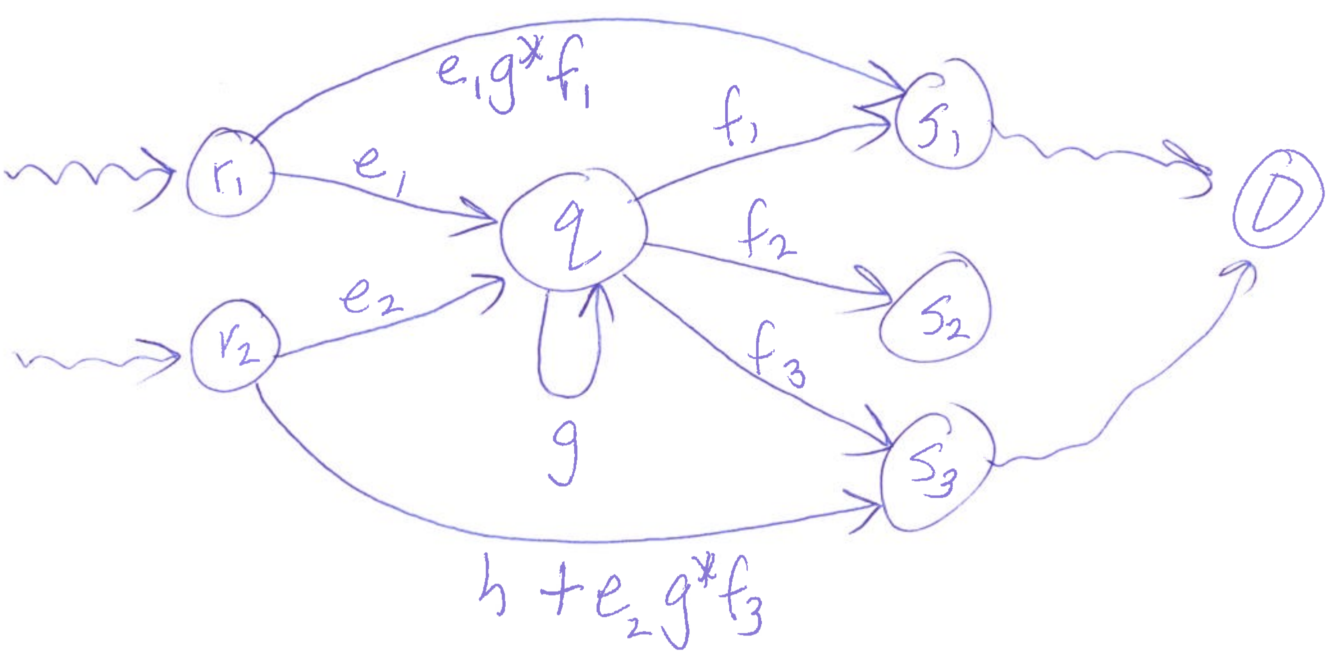
– Result is G_i // $L(G_i) = L(G_{i-1})$

end-for

4. Let G_n be the last GNFA constructed in step 3. G_n has no intermediate states.

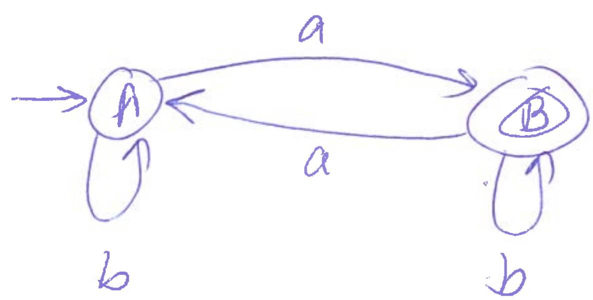
$$L(r) = L(G_n) = L(G_{n-1}) = \cdots = L(G_0) = L(N)$$

<u>State elimination</u>: Eliminate q.



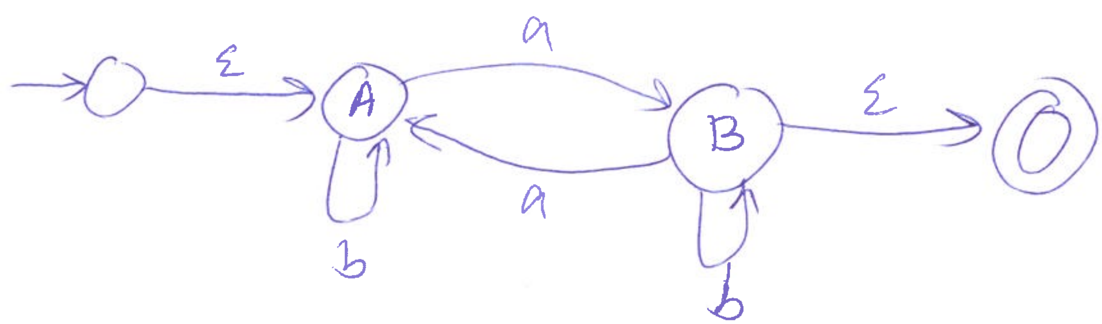
h is the regex already existing from r_2 to s_3 before q is eliminated.

Ex: N:

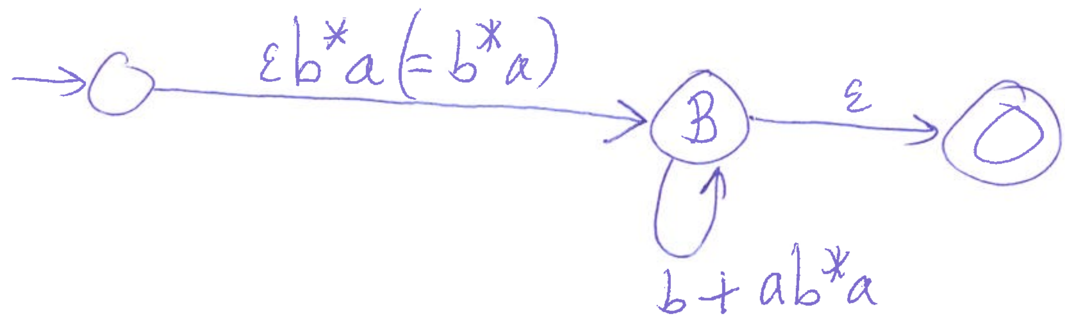


$L(N) = \{w \in \{a,b\}$: w has an odd # of a's$\}$

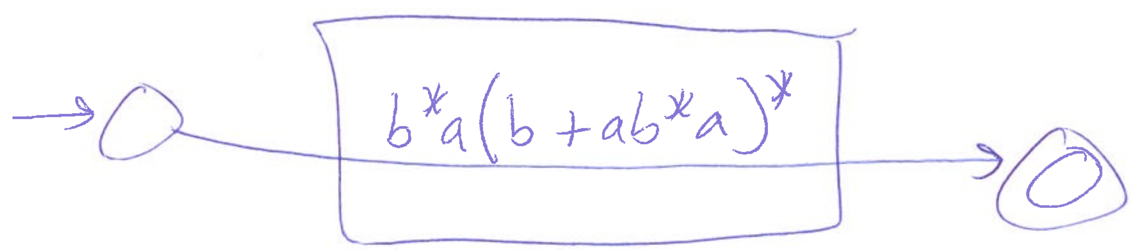
1. G_0



G_1

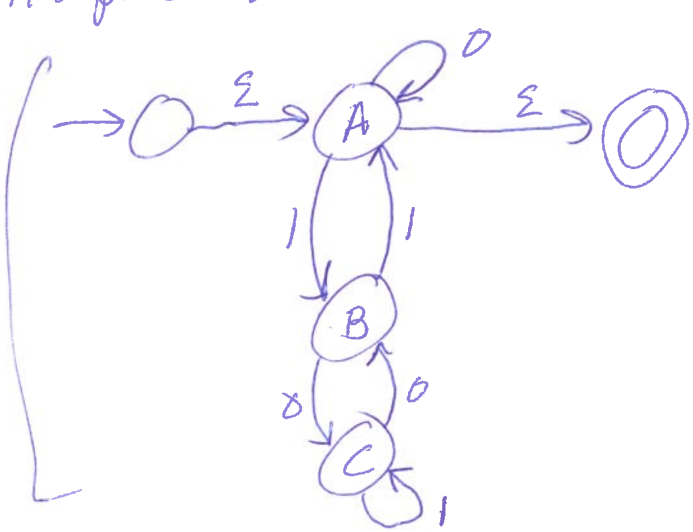


G_2



Binary multiples of 3:

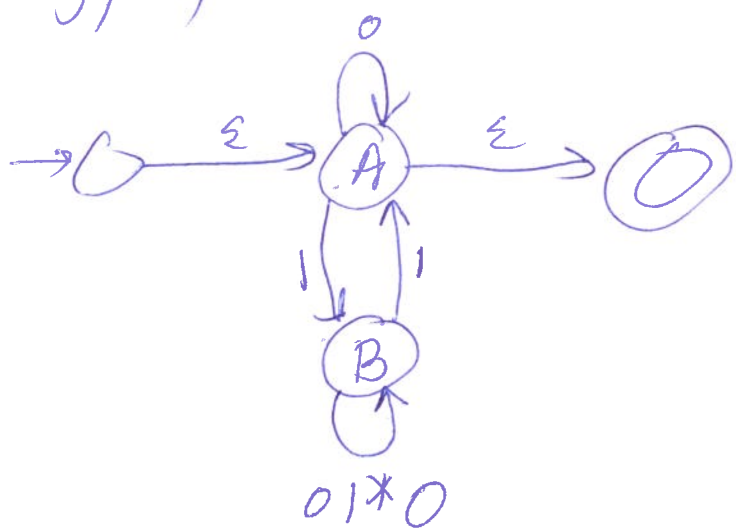
clean version of the original DFA



Strategy: choose a state to eliminate the requires the fewest bypasses.
("Lazy strategy")

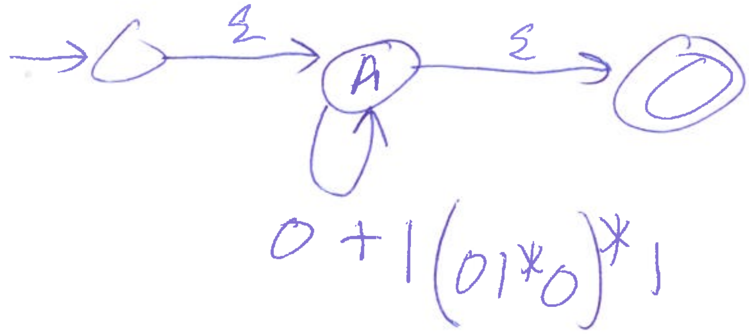
Elim C:

G_1:



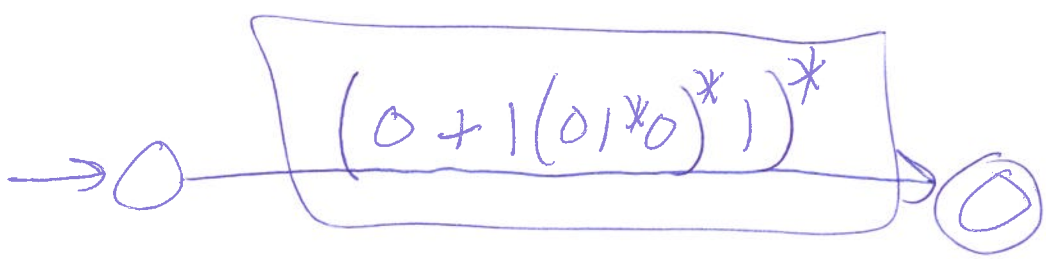
Elim B:

G_2:



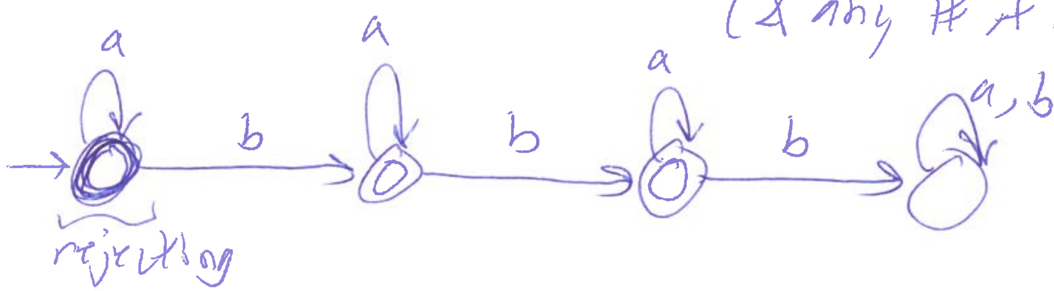
Elim A:

G_3



Ex: $\Sigma = \{a, b\}$ $L = \{w : w$ has one or two b's (& any # of a's)$\}$

DFA:



Try this yourself.

Def: For alphabet Σ, let REX_Σ be the set of all regexes over Σ.

Def: A GNFA is a 5-tuple $\langle Q, \Sigma, \delta, q_0, F \rangle$ where Q, Σ, q_0, F are the same as with an ε-NFA/NFA/DFA, and

$$\delta : Q \times Q \longrightarrow REX_\Sigma$$

Idea: $\delta(q,r)$ = regex labeling the unique edge from q to r.

$q \xrightarrow{e} r$ means $\delta(q,r) = e$

q (no arrow) r means $\delta(q,r) = \emptyset$

[same as $q \xrightarrow{\emptyset} r$]

cannot go from q to r directly

Def. Let $G = \langle Q, \Sigma, \delta, q_0, F \rangle$ be a GNFA and let w be a string over Σ.

A complete comp. path of G on w is a sequence $s_0, s_1, \ldots, s_k \in Q$ such that there exist strings $w_1, w_2, \ldots, w_k \in \Sigma^*$ $(k \geq 0)$ ~~and~~ such that

1. $w = w_1 \cdots w_k$
2. $s_0 = q_0$
3. $\forall i, 1 \leq i \leq k,$

$$w_i \in L(\delta(s_{i-1}, s_i))$$

The path ends in s_k.

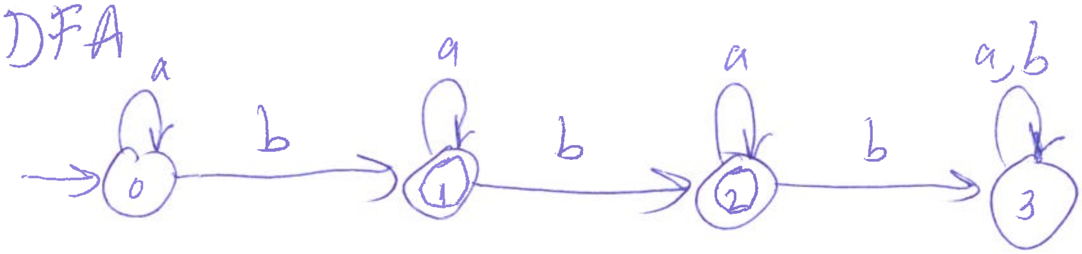
G accepts w if $\exists$ complete comp. path on w ending in an accepting state.

CSCE 355
2/13/2023

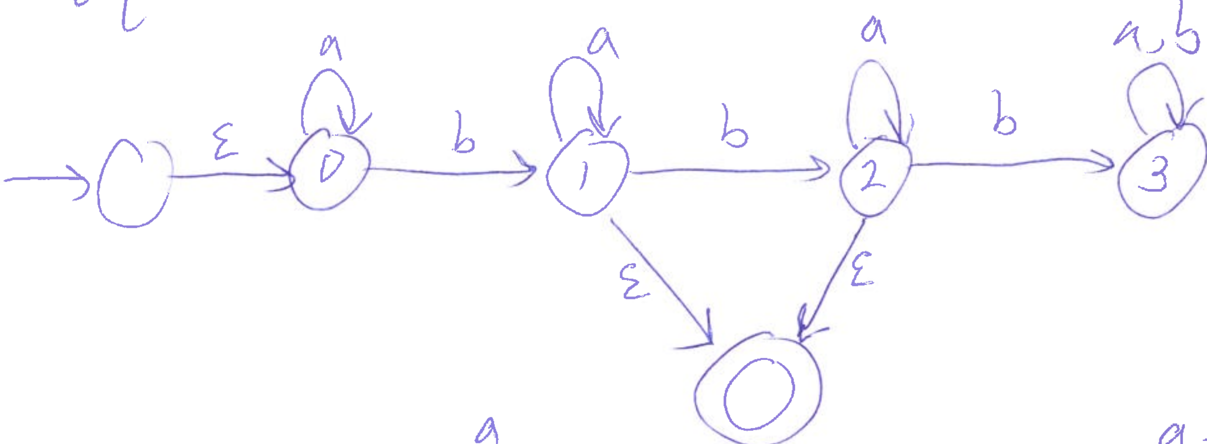
<u>Ex</u>: state elimination method (1)

$L = \{ w \in \{a,b\}^* : w \text{ has one or two } b\text{'s } (\& \text{ any \# of } a\text{'s}) \}$

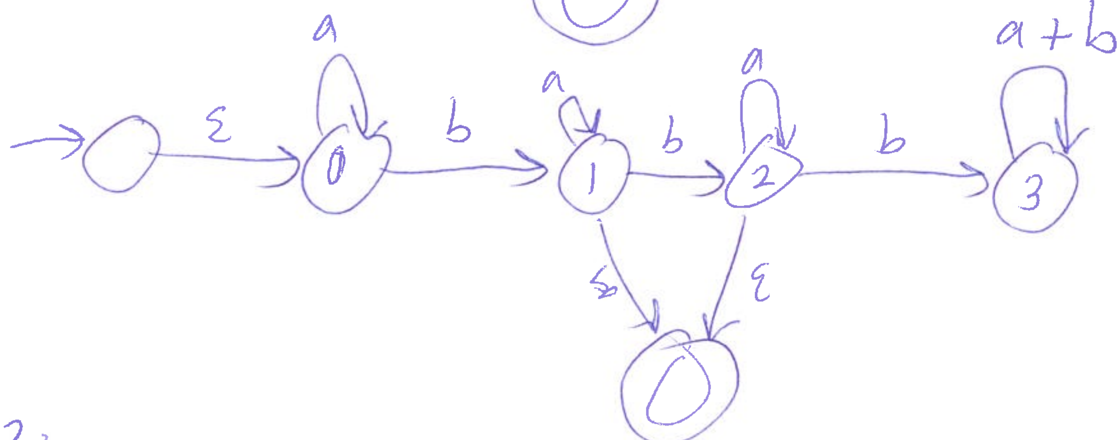
DFA



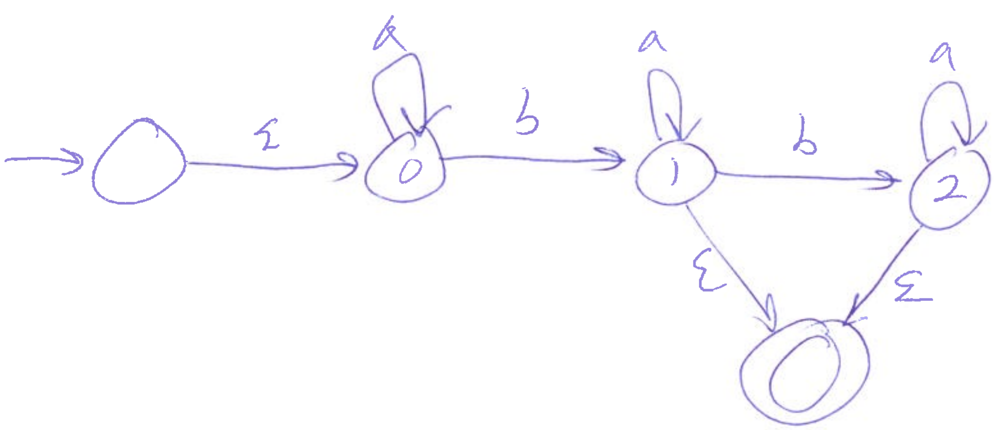
Equiv clean ε-NFA:



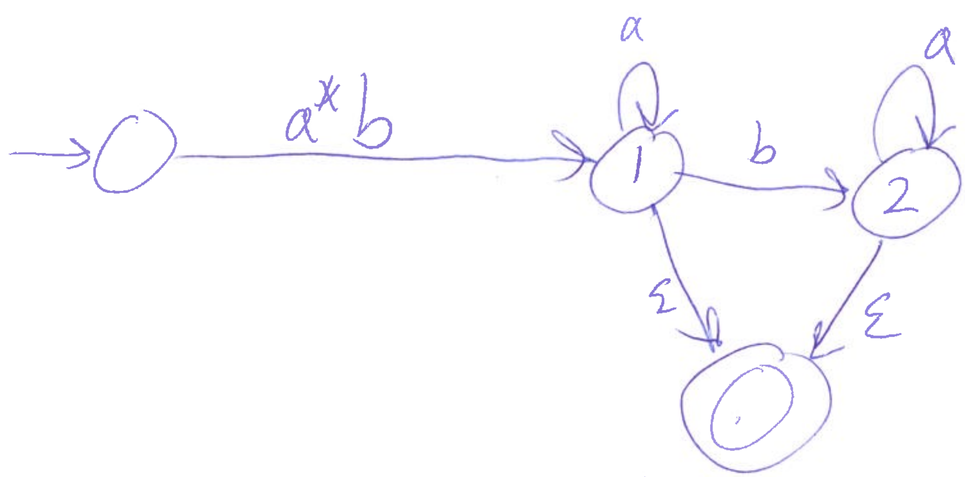
G_x



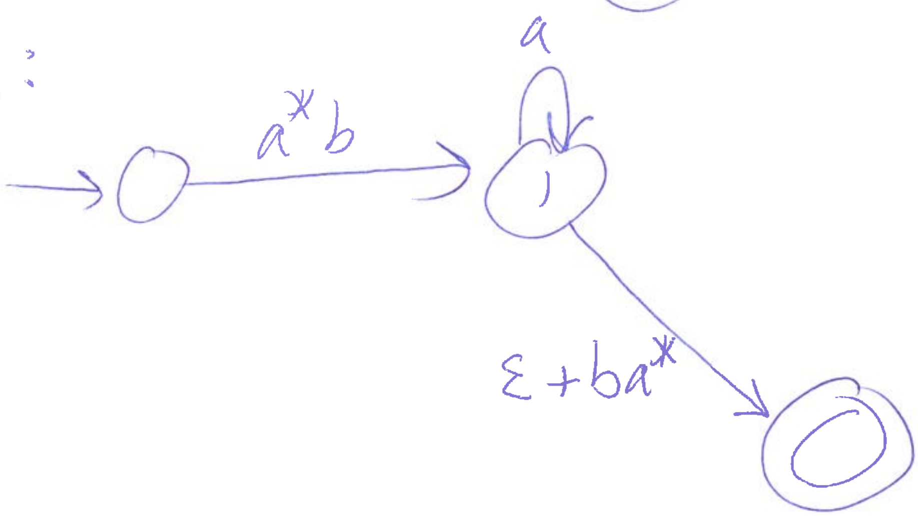
Elim 3:



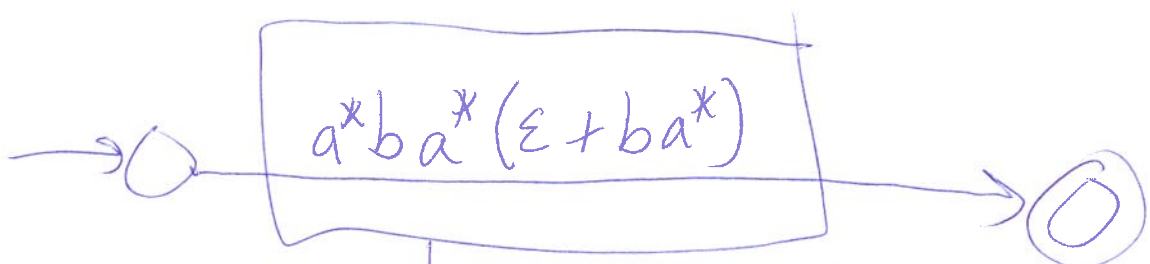
Elim 0: (could elim 2 instead)



Elim 2:



Elim 1:



$\equiv \boxed{a^*ba^*(ba^*)?}$

Thm (proved!)

~~A language~~ The regular languages are characterized by any the following:

- DFA recognition
- NFA "
- ε-NFA "

- GNFA ''
- regexes

Closure properties of REG_Σ, the class of regular languages over Σ

Ex: string reversal

Def: Let $w = w_1 w_2 \cdots w_n$ (each $w_i \in \Sigma$) $n = |w|$

The reversal w^R of w is $w^R := w_n \cdots w_2 w_1$.

Note: If $x, y \in \Sigma^*$, then $(xy)^R = y^R x^R$ and $(x^R)^R = x$

Def: Let $L \subseteq \Sigma^*$ be a language. Define

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

the reversal of L.

Note: $\underline{(L_1 L_2)^R} = \{xy : x \in L_1 \ \& \ y \in L_2\}^R$

$$= \{(xy)^R : x \in L_1 \ \& \ y \in L_2\}$$

$$= \{y^R x^R : y \in L_2 \ \& \ x \in L_1\}$$

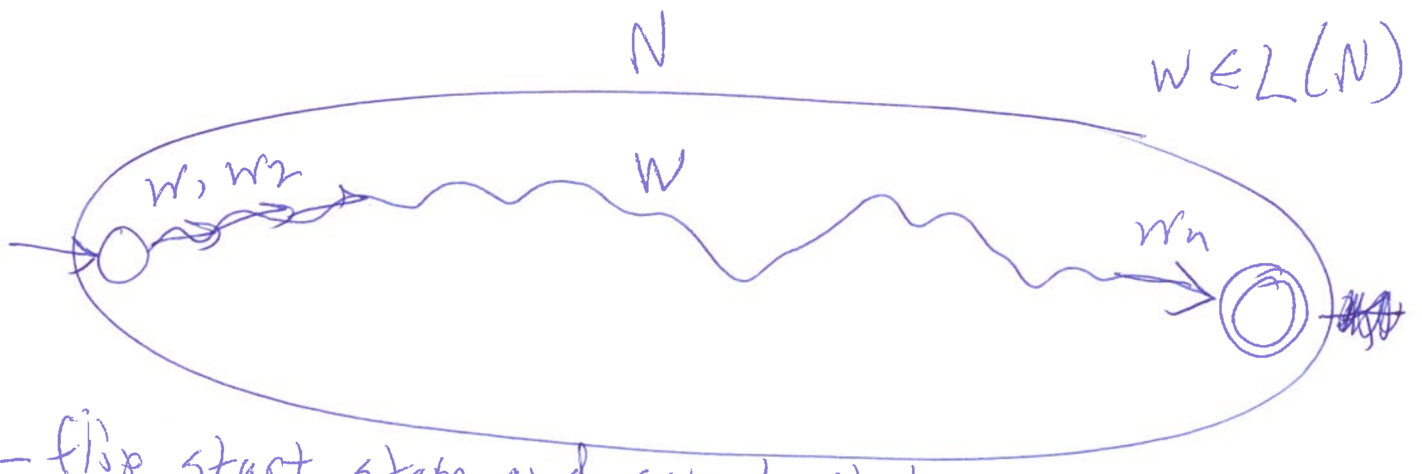
(4)

$$= \{y^R : y \in L_2\}\{x^R : x \in L_1\}$$
$$= \underline{L_2^R L_1^R}$$

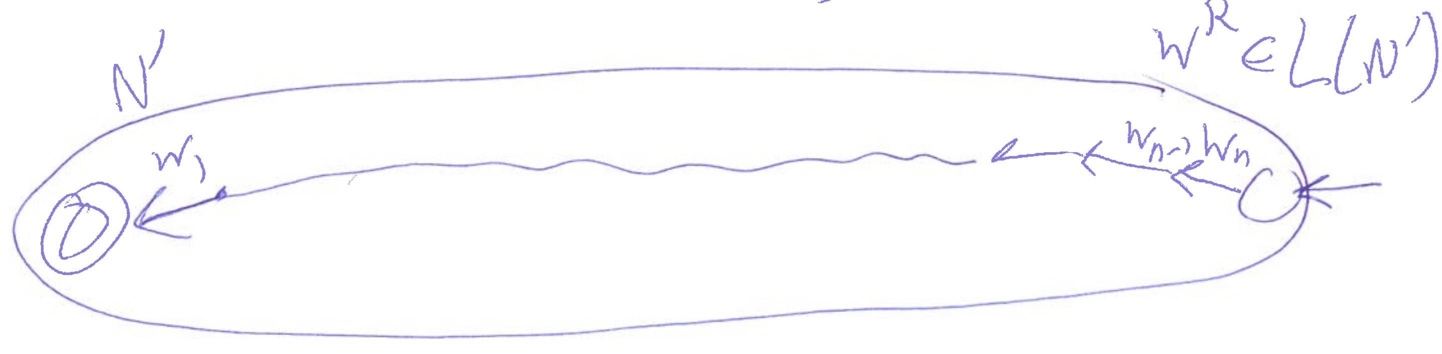
Also: $(L^R)^R = L$.

<u>Prop</u>: If L is regular, then L^R is regular.

<u>Proof 1</u>: Given a clean ε-NFA N, find an ~~(clean~~ ε-NFA N' such that $L(N') = L(N)^R$, as follows:



— flip start state and accept state,
– reverse all arrows (transitions)



"Clear": If N accepts a string w, then N' accepts w^R (follow reverse in N' of accepting path in N, to accept w^R).

$$\therefore L(N)^R \subseteq L(N')$$

WTS $L(N') \subseteq L(N)^R$ $\therefore L(N') = L(N)^R$

But $L(N')^R \subseteq L(\underbrace{N''}_{=N}) = L(N)$

$\therefore \underbrace{\left(L(N')^R\right)^R}_{= L(N')} \subseteq L(N)^R$ QED. //

apply R to both sides

Proof 2: Given any regex r, ~~find~~ define a regex r' such that $L(r') = L(r)^R$.

Induction on ~~the~~ the syntax of r:

	r	r'	
	$\emptyset$	$\emptyset$	Base cases (atomic regexes)
$(a \in \Sigma)$	a	a	

s, t regexes over Σ

$s+t$	$s'+t'$
st	$t's'$
s^*	$(s')^*$

$\cancel{*}$ ← only change

Pf of correctness based on how R-operator works with lang. concat & union.

Ex: $[(abc^* + ba)^*]'$

$= (c^*ba + ab)^*$

Def: w a string over $\Sigma = \{a, b, c\}$

DOUBLE-a(w) = string obtained by replacing each occurrence of "a" in w with "aa"

ex: DOUBLE-a(bacaab) = b$\underline{aa}$c$\underline{aa}\underline{aa}$b

Given $L \subseteq \Sigma^*$, define

$$\text{DOUBLE-a}(L) = \{\text{DOUBLE-a}(w) : w \in L\}$$

Prop: If L is regular, then DOUBLE-a(L) is regular.

Proof 1: Given an NFA (or DFA or ε-NFA) N,

Find an NFA (ε-NFA) N' such that
$L(N') = \text{DOUBLE-}a(L(N))$:

For every a-transition in N, say,

replace it with

Proof 2: Given a regex r over $\{a,b,c\}$,
convert to a regex r' such that
$L(r') = \text{DOUBLE-}a(L(r))$:

r	r'
$\emptyset$	$\emptyset$
a	aa
b	b
c	c
$s+t$	$s'+t'$
st	$s't'$
s^*	$(s')^*$

Closure under string homomorphisms

<u>Def</u>: Let Σ, Γ be alphabets.

A <u>string</u> <u>homomorphism</u> (Σ to Γ) is a map $\varphi : \Sigma^* \longrightarrow \Gamma^*$

such that
$$\varphi(\overbrace{xy}^{\Sigma^*}) = \underbrace{\varphi(x)\varphi(y)}_{\in \Gamma^*}$$
for any $x, y \in \Sigma^*$.

<u>Fact</u>: $\varphi(\varepsilon) = \varepsilon$

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

CSCE 355
2/15/2023

Closure under homomorphic images and inverse homomorphic images.

Recall: Σ, Γ alphabets. $\varphi : \Sigma^* \to \Gamma^*$ is a string homomorphism if

$$\varphi(xy) = \varphi(x)\varphi(y)$$

Prop: If φ is a str. homo., then $\varphi(\varepsilon) = \varepsilon$.

Proof: $\underline{\varphi(\varepsilon)} = \varphi(\varepsilon\varepsilon) = \underline{\varphi(\varepsilon)}\,\underline{\varphi(\varepsilon)}$

$\therefore |\varphi(\varepsilon)| = 2|\varphi(\varepsilon)|$

$\therefore \varphi(\varepsilon) = 0$ $\quad\therefore \varphi(\varepsilon) = \varepsilon$. //

Prop: φ is completely determined by how it maps strings of length 1:

Proof: $w \in \Sigma^*$: $w = w_1 w_2 \cdots w_n$ $\quad (w_i \in \Sigma)$

$$\varphi(w) = \varphi(w_1 w_2 \cdots w_n) = \varphi(w_1)\varphi(w_2 \cdots w_n)$$
$$= \varphi(w_1)\varphi(w_2)\varphi(w_3 \cdots w_n) = \cdots = \varphi(w_1)\varphi(w_2)\cdots\varphi(w_n) \quad //$$

Converse: Any map $\tilde{\varphi} : \Sigma \to \Gamma^*$

extends (uniquely) to a string homo.

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

[φ agrees with $\tilde{\varphi}$ on Σ]:

$$\varphi(w_1 \cdots w_n) := \tilde{\varphi}(w_1) \cdots \tilde{\varphi}(w_n).$$

Ex: $\Sigma = \{a, b, c\}$, $\Gamma = \{0, 1\}$

$$\tilde{\varphi} : \begin{array}{l} a \longmapsto 010 \\ b \longmapsto 11 \\ c \longmapsto \varepsilon \end{array}$$

$$\varphi(aabcbca) = \underline{010}\,\underline{010}\,\underline{11}\,\underline{11}\,\underline{010}$$

Def: Let $L \subseteq \Sigma^*$ be a lang. and let $\varphi : \Sigma^* \longrightarrow \Gamma^*$ be a str. homo. Define

$$\varphi(L) := \{\varphi(w) : w \in L\} \subseteq \Gamma^*$$

(image of L under φ).

Let $M \subseteq \Gamma^*$ be a lang. Define

$$\varphi^{-1}(M) = \{w \in \Sigma^* : \varphi(w) \in M\}$$

(inverse image of M under φ).

Fix a string homo. $\varphi : \Sigma^* \to \Gamma^*$ 3

Thm: If $L \subseteq \Sigma^*$ is regular, then $\varphi(L)$ is regular.

Proof: Given a regex r over Σ, show how to transform r into a regex r' over Γ such that $\boxed{L(r') = \varphi(L(r))}$.

r' defined by induction (recursion) on syntax of r:

r	r'	
$\emptyset$	$\emptyset$	
a $(a \in \Sigma)$	$\boxed{\varphi(a)}$	string over Γ interpreted as a regex
$s + t$	$s' + t'$	image of union is union of images
st	$s't'$	because φ ~~resp~~ preserves concatenations
s^*	$(s')^*$	same reason

Proof of correctness omitted. //

~~(4)~~ Last time example: L over $\{a,b,c\}$

$$\text{DOUBLE-a}(L) = \{w : w \text{ is obtained from a string in } L \text{ by replacing each "a" by "aa"}\}$$

L reg $\Rightarrow$ DOUBLE-a(L) is regular.

This is a special case of the theorem just proved:

$$\text{DOUBLE-a}(L) = \varphi(L)$$

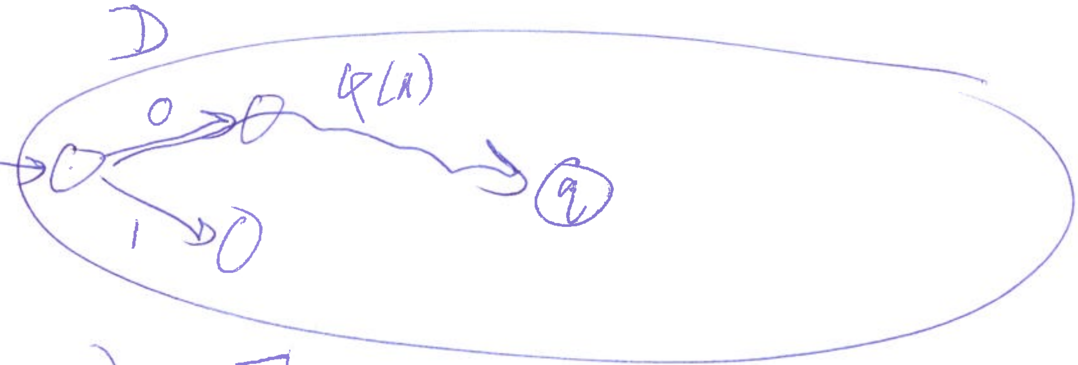
where $\varphi : \Sigma^* \longrightarrow \Sigma^*$ is the str. homo. that maps

$$a \longmapsto aa$$
$$b \longmapsto b$$
$$c \longmapsto c.$$

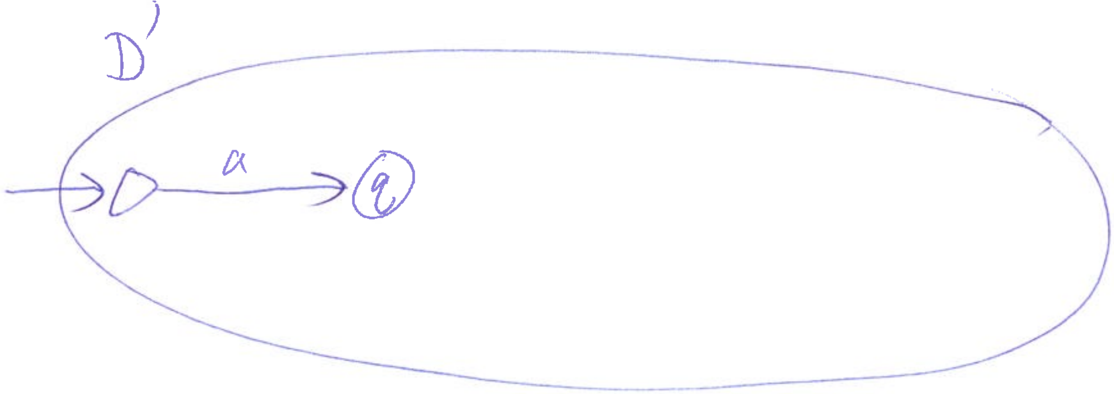
<u>Thm</u>: $\varphi : \Sigma^* \longrightarrow \Gamma^*$ str. homo., $M \subseteq \Gamma^*$ any lang. over Γ. If M is regular, then $\varphi^{-1}(M)$ $(\subseteq \Sigma^*)$ is regular.

<u>Proof</u>: Let D be a DFA such that $M = L(D)$.

Idea:



$\varphi(abca) \in M_0^? \Leftrightarrow M$ accepts $\varphi(abca) = \varphi(a)\varphi(b)\varphi(c)\varphi(a)$



If $D = \langle Q, \Gamma, \delta, q_0, F \rangle$, define

$$D' := \langle Q, \Sigma, \delta', q_0, F \rangle \text{ where}$$

$\forall a \in \Sigma, \forall q \in Q,$

$$\delta'(q, a) = \hat{\delta}(q, \varphi(a)).$$

Can Prove by induction on length of $w \in \Sigma^*$ that $\hat{\delta}'(q, w) = \hat{\delta}(q, \varphi(w))$

It follows that D' accepts w iff D accepts $\varphi(w)$.

$\therefore L(D') = \varphi^{-1}(L(D))$. //

Ex: $\Sigma = \{a, b, c\}$, $\Gamma = \{0, 1\}$

$\varphi : \Sigma^* \to \Gamma^*$ str. homo. s.t.

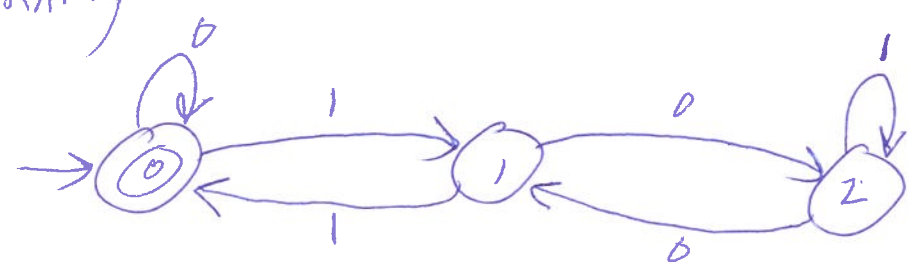
$$\varphi(a) = 010$$
$$\varphi(b) = 11$$
$$\varphi(c) = \varepsilon$$

Regex r over $\{a, b, c\}$ $\qquad \varepsilon = \emptyset^*$

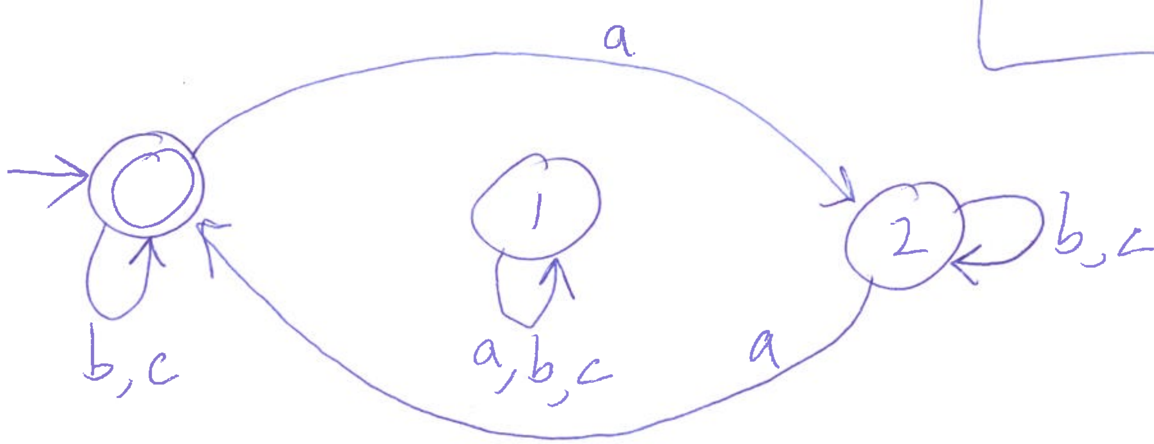
$$\left[(ab^* + (cab)^* + \varepsilon)^* \right]' \qquad \varphi(\varepsilon) = \varepsilon$$

$$= (010(11)^* + (01011)^* + \varepsilon)^*$$

$M \subseteq \Gamma^*$ contains the multiples of 3 in binary

$D =$ 

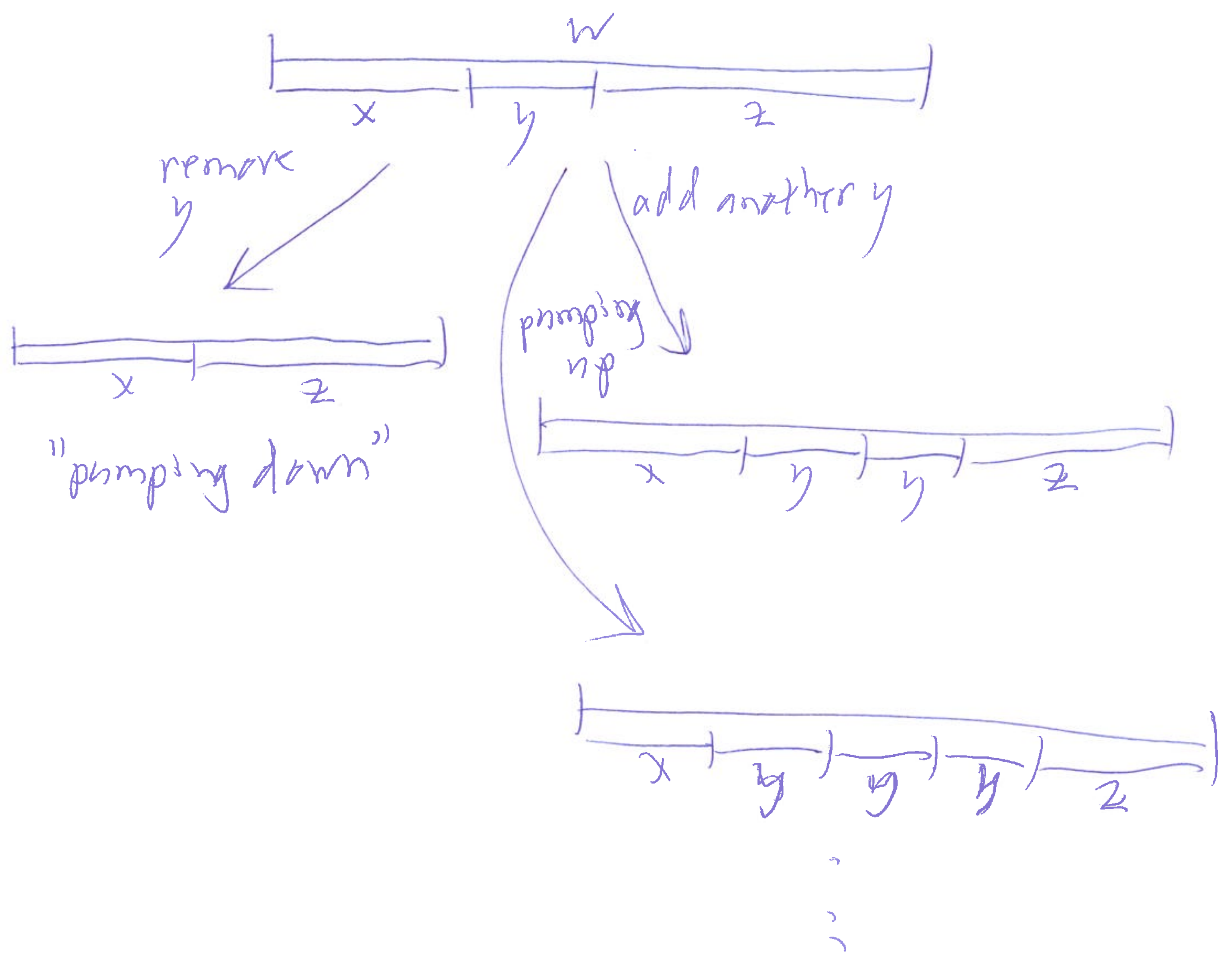
$a \mapsto 010$
$b \mapsto 11$
$c \mapsto \varepsilon$

$D' :=$ 

Pumping Lemma for regular languages

Let w be a string, and let y be a substring of w

"Pumping on y" means replacing y with any number of copies of y (incl. zero):



<u>Def</u>: Let $L \subseteq \Sigma^*$. We say that L is <u>pumpable</u> if

$\exists p > 0$ (the "pumping length")

$\forall s \in L$ such that $\underbrace{|s| \geq p}_{s \text{ is "sufficiently long"}}$

$\exists x, y, z \in \Sigma^*$ such that

1) $s = xyz$ (y is a substring of s)
2) $|xy| \leq p$ (y is "close to the beginning of s")
3) $|y| > 0$ ($y \neq \varepsilon$)

$\forall i \geq 0$, $xy^i z \in L$.

<u>Pumping Lemma</u>: Every regular language is pumpable.

<u>Proof idea</u>: Let L be regular, and let D be a DFA recog. L.

Let p be the # of states of D

Any $s \in L$, $|s| \geq p$. $s = s_1 \cdots s_p \cdots$