① CSCE 355 1/9/23

Spring 2023

Foundations of Computing

https://cse.sc.edu/~fenner/csce355

fenner@cse.sc.edu

Computation takes input(s) and produces an output.

<u>Def</u>: An <u>alphabet</u> is any nonempty finite set. If Σ is an alphabet, we call the elements of Σ <u>symbols</u>, <u>letters</u>, or <u>characters</u>.

~~A~~ Given an alphabet Σ, a <u>string</u> over Σ is any finite sequence of symbols from Σ.

<u>Ex</u>: $\Sigma = \{a, b, c\}$

② Some strings over Σ:

- aab
- bca
- a
- ccccccc
- baa
- ba
- ε (the empty string)

$\neq$

$\neq$

ε is never a symbol of the alphabet. It stands for the empty string.

<u>Def</u>: The <u>length</u> of a string is the # of symbols making up the string including duplicates.

If x is a string, we let $|x|$ denote the length of x. So $|aab| = 3$, etc.

String concatenation: If x & y are strings, The <u>concatenation</u> of x with y, written xy is the string y appended to x.

Ex: $x = aab$

$y = ca$

$xy = aabca$

$yx = caaab$

Concat is associative:

$$(xy)z = x(yz) = xyz$$

[x, y, z are strings]

More generally, $x_1 x_2 x_3 \cdots x_k$

x^n [x string, $n \geq 0$ integer]

$x^n := \underbrace{xx\cdots x}_{n \text{ times}}$ [$x^0 := \varepsilon$ by convention
$x^1 = x$]

Special case: (1) $\Sigma = \{0,1\}$ binary alphabet

Strings over $\{0,1\}$ are binary strings.

(2) $\Sigma = \{0\}$ unary alphabet (unary strings)

unary strings $\varepsilon, 0, 00, 000, \ldots, 0^n, \ldots$

④ <u>Def</u>: Σ alphabet. The set of all strings over Σ (incl. ε) is denoted Σ^*

$$\{0,1\}^* = \{\underbrace{\varepsilon}_{0}, \underbrace{0, 1}_{1}, \underbrace{00, 01, 10, 11}_{2}, \underbrace{000, \ldots}_{3}\}$$

"length-first lexicographical order"

- Symbols from Σ are identified with strings of length 1.

$$|\varepsilon| = 0 \quad \text{(no other string has length 0)}$$

strings x, y $\quad |xy| = |x| + |y|$

Induction on string length

Basic principle: For any string $x \in \Sigma^*$

Either

not both

$x = \varepsilon$ — base case

or

there exist unique $y \in \Sigma^*$ and $a \in \Sigma$ such that $x = ya$.

a is the <u>last</u> <u>symbol</u> of x, ~~and y is the~~ principal and y is the principal prefix of x.

Note: $|y| = |x| - 1 < |x|$

Note: $\varepsilon x = x \varepsilon = x$ (x any string)

ε is ~~an~~ the identity under concatenation.

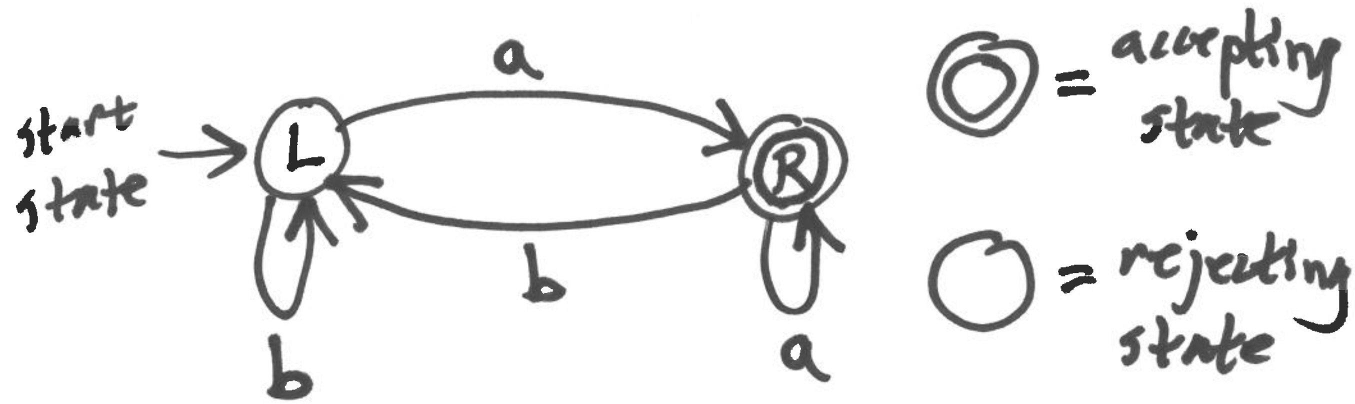
Finite Automata (model of computation)

An automaton will take a string as input, read it left to right, symbol by symbol, and at the end either accept or reject (the input)

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

Automaton that accepts a string if and only if it ends in a (has a as the last symbol).

6

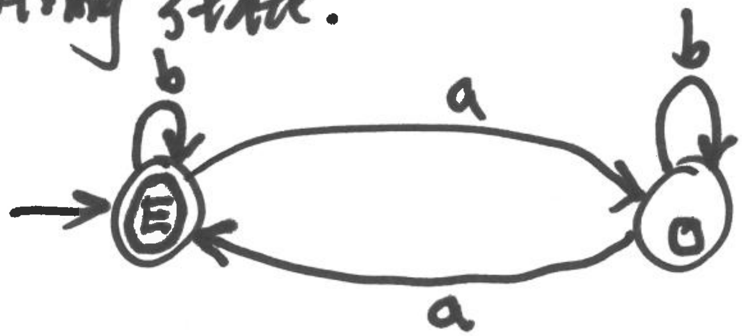


Input : abbbaa

↑ ↑ ↑ ↑ ↑ ↑ ↑

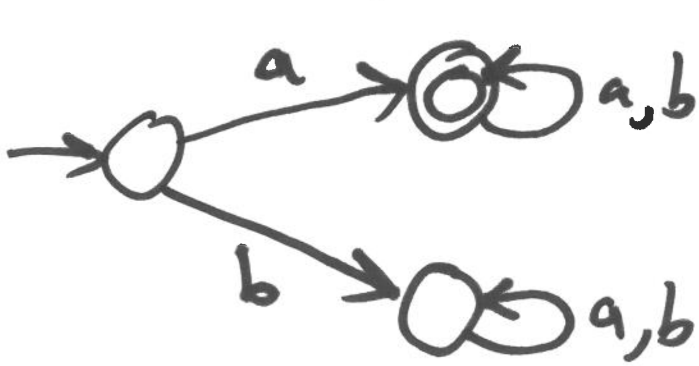
L R L L L R R

Automaton accepts (by def) just when its last state (after the whole input) is an accepting state.



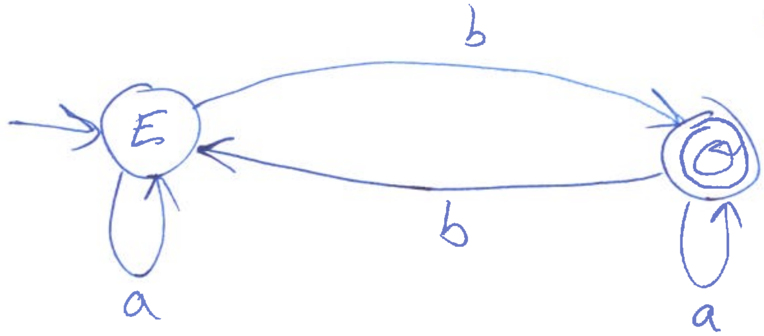
Accepts strings with even # of a's

Accepts iff 1st symbol is a



① CSCE 355 1/11/2023

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



transition diagram

Def: A deterministic finite automaton (DFA) is a 5-tuple $\langle Q, \Sigma, \delta, q_0, F \rangle$ where

- Q is a finite set (elements of Q are states)
  [Ex: $Q = \{E, O\}$]
- Σ is an alphabet (the input alphabet)
  [Ex: $\Sigma = \{a, b\}$]
- $q_0 \in Q$ (the start state) [Ex: $q_0 = E$]
- $F \subseteq Q$ (elements of F are the accepting states; states not in F are rejecting states)
  [Ex: $F = \{O\}$]
- $\delta : Q \times \Sigma \to Q$ (the transition function)

$$\delta(E,a)=E \qquad \delta(E,b)=O$$
$$\delta(O,a)=O \qquad \delta(O,b)=E$$

As a table:

$\rightarrow$ = (start state)

$*$ = (accepting state)

	a	b
$\rightarrow E$	E	O
$* O$	O	E

tabular form

$$\text{transition diagram} \underset{\text{convert}}{\rightleftarrows} \text{tabular form}$$

Recall: Σ^* is the set of all strings over Σ.

Given a DFA $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ want to extend the def of δ to apply to strings.

Ex:

$$\textcircled{q} \xrightarrow{a} \textcircled{r} \xrightarrow{b} \textcircled{s}$$

$\delta(q,a)=r \qquad \delta(r,b)=s$

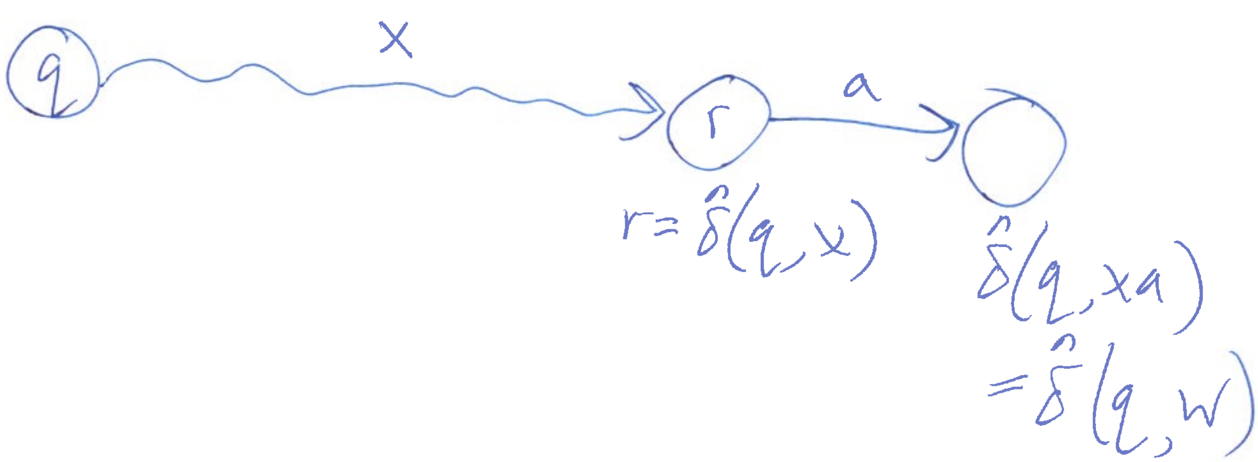
$$\hat{\delta}(q,ab) = s = \delta(\delta(q,a),b) = \delta(r,b)$$

(3) <u>Def</u>: Given A as above we define the extended <u>transition function</u> $\hat{\delta} : Q \times \Sigma^{*} \to Q$ inductively as follows:

- $\hat{\delta}(q, \varepsilon) = q \qquad (\forall q \in Q)$

- For any $q \in Q$ and string $w \neq \varepsilon$, let $w = xa$, where x is the principal prefix of w and a is the last symbol of w.

$$\hat{\delta}(q, w) = \delta(\underbrace{\hat{\delta}(q, x)}_{r}, a)$$



<u>Prop</u>: $\forall q \in Q, \forall a \in \Sigma, \ \hat{\delta}(q, a) = \delta(q, a)$

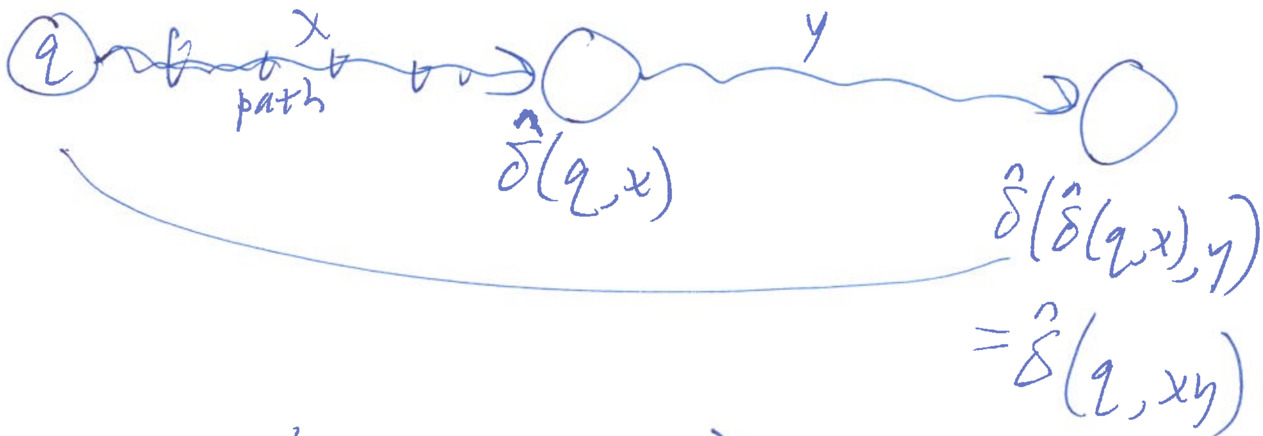
<u>Proof</u>: $\hat{\delta}(q, a) = \hat{\delta}(q, \varepsilon a)$

$$= \delta(\underbrace{\hat{\delta}(q, \varepsilon)}_{q}, a) = \delta(q, a) \ \checkmark$$

$\boxtimes$

④ **Prop:** $\forall q \in Q$, $\forall x, y \in \Sigma^*$,

$$\hat{\delta}(q, xy) = \hat{\delta}(\hat{\delta}(q, x), y)$$



Def: Let $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ be a DFA and $w \in \Sigma^*$ a string over Σ.

A *accepts* w means $\hat{\delta}(q_0, w) \in F$. [otherwise, A *rejects* w]

Equivalently, suppose $w = w_1 w_2 \cdots w_n$ ($n \geq 0$ and $w_i \in \Sigma$).

The *computation* (*computation path*) (*computational trace*) is the sequence of states $s_0, s_1, \ldots, s_n \in Q$ that A goes through reading w

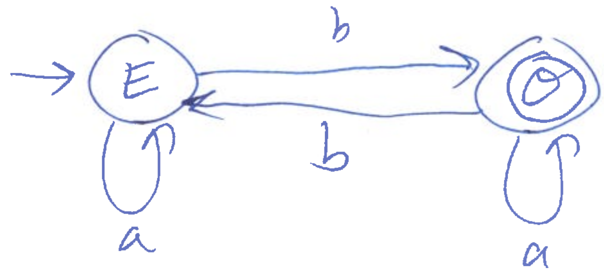
That is

- $s_0 = q_0$
- for all $1 \leq i \leq n$, $s_i = \delta(s_{i-1}, w_i)$

⑤ we say the computation ends in s_n.

A accepts w iff its computation on input w ends in an accepting state $(s_n \in F)$.

Ex:



Input: ababb

computation: E, E, O, O, E, O

$$E \xrightarrow{a} E \xrightarrow{b} O \xrightarrow{a} O \xrightarrow{b} E \xrightarrow{b} O$$

Fix an alphabet Σ.

Def: A language over Σ is any subset of Σ^* (any set of strings over Σ)

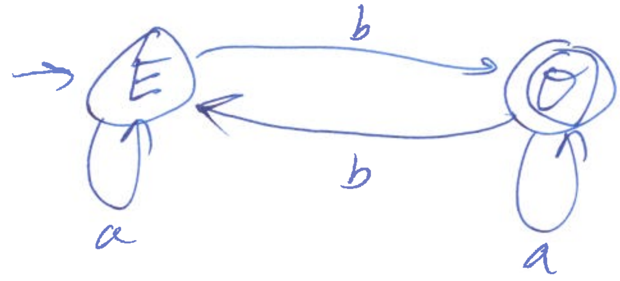
Def: Given a DFA A with input alphabet Σ, The language $L(A)$ recognized by A is

$$L(A) := \{x \in \Sigma^* : A \text{ accepts } x\}$$

A recognizes $L(A)$.

6

Ex: $A =$

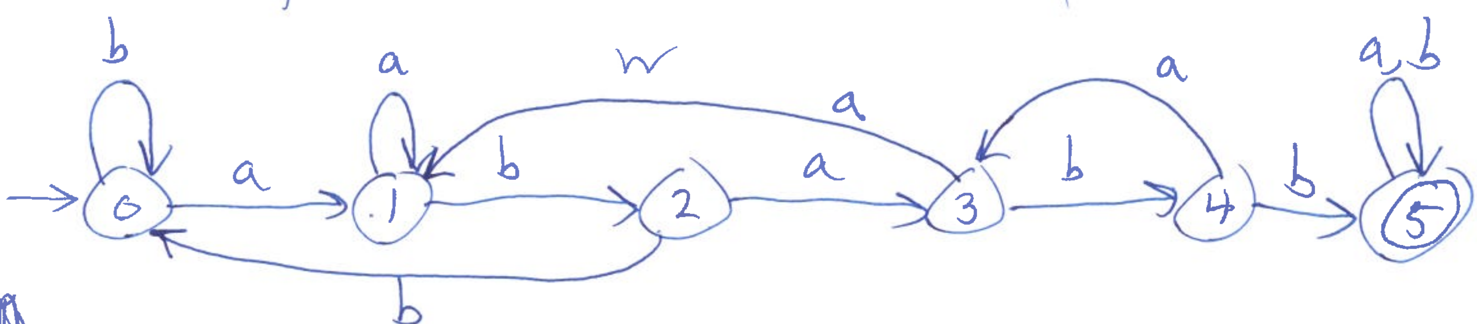


$L(A) = \{ \underline{w} \in \{a,b\}^* : w$ has an odd number of b's $\}$

abb $\notin L(A)$ rejected
baa $\in L(A)$ accepted

DFA examples: $\Sigma = \{a, b\}$

Want a DFA that accepts a string w iff it has $\underline{ababb}$ (search string) as a substring



<u>Idea:</u> DFA is in state i iff ~~the~~ it has read a prefix of the search string of length i (but not greater)

---

⑦ DFA ~~that~~ (input alphabet $\{0,1\}$) that accepts a binary string iff it represents a multiple of 3 in binary. (ε represents 0 by convention)

$0, \varepsilon = 0$

$1 = 1$

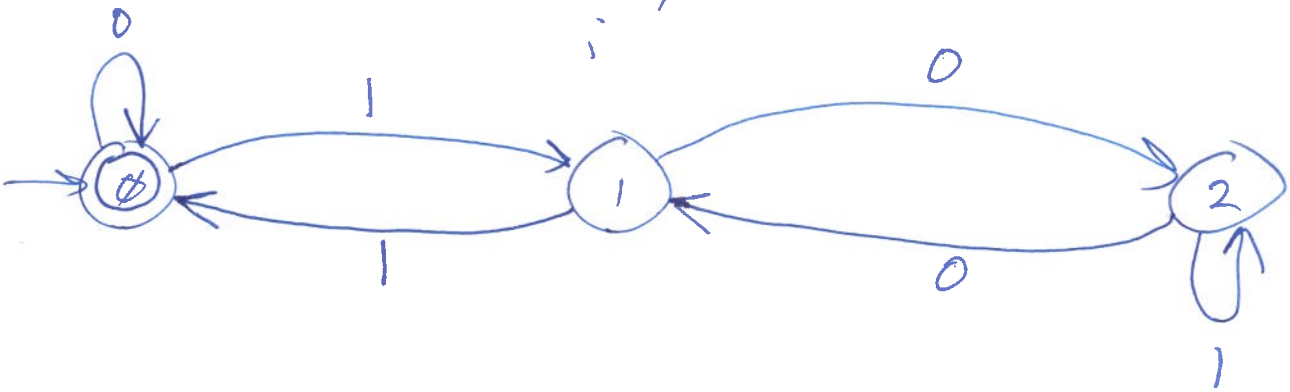
$10 = 2$

$11 = 3$

$100 = 4$

$\vdots$

1,10001



CSCE 355
1/18/2023

Def: Fix an alphabet Σ. Let $L \subseteq \Sigma^*$ be a language over Σ. L is <u>regular</u> if there exists a DFA recognizing L ($L = L(A)$ for some DFA A).

Def: For alphabet Σ, REG_Σ is the class of all regular languages over Σ.

$$REG_\Sigma := \{L \subseteq \Sigma^* : L \text{ is regular}\}$$
$$= \{L(A) : A \text{ is a DFA with input alphabet } \Sigma\}$$

Def: $L \subseteq \Sigma^*$. The complement of L (in Σ^*) is

$$\overline{L} := \{w \in \Sigma^* : w \notin L\} = \boxed{\Sigma^*} \setminus L$$

Prop: The complement of a regular language is regular. (REG_Σ is closed under complements.)

Proof: By construction: Given any DFA ~~A~~

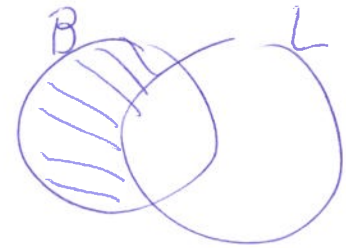
$A = \langle Q, \Sigma, \delta, q_0, F \rangle$

define the DFA

$\neg A := \langle Q, \Sigma, \delta, q_0, \underline{Q \setminus F} \rangle$

("complement construction")

$B \setminus L := \{x : x \in B \;\&\; x \notin L\}$



Claim that $L(\neg A) = \overline{L(A)}$.

Given any $w \in \Sigma^*$, let $q := \hat{\delta}(q_0, w)$ (same in $\neg A$ as in A)

$$A \text{ accepts } w \iff q \in F$$

def of acceptance

$$\iff q \notin Q \setminus F$$

$$\iff \neg A \text{ rejects } w$$

$\therefore L(\neg A) = \{w : A \text{ rejects } w\} = \overline{L(A)}$

$\therefore$ If a lang is regular, so is its complement. //

Prop: REG_Σ is closed under intersection.

(The intersection of two regular languages is regular.)

__Proof:__ By construction. Let ~~A B~~

$$A := \langle Q_A, \Sigma, \delta_A, \underline{q_{0,A}}, F_A \rangle$$

and

$$B := \langle Q_B, \Sigma, \delta_B, \underline{q_{0,B}}, F_B \rangle$$

be any DFAs with input alphabet Σ.

Construct a DFA

$$C := \underline{A \wedge B} := \langle Q, \Sigma, \delta, q_0, F \rangle, \text{ where}$$

$$Q := Q_A \times Q_B \;(= \{(q,r) : q \in Q_A \,\&\, r \in Q_B\}),$$

$$q_0 := (\underline{q_{0,A}}, \underline{q_{0,B}})$$

$$F := \{(q,r) : q \in F_A \overset{\wedge}{\&} r \in F_B\} = F_A \times F_B$$

and, for every $q \in Q_A$ and $r \in Q_B$, and every $a \in \Sigma$,

$$\delta((q,r),a) := (\delta_A(q,a), \delta_B(r,a)).$$

__Claim:__ For every $q \in Q_A$, $r \in Q_B$, $w \in \Sigma^*$,

$$\underline{\hat{\delta}((q,r),w) = (\hat{\delta}_A(q,w), \hat{\delta}_B(r,w))}$$

[WTS: $L(A \wedge B) = \cancel{L(A) \wedge L(B)}\; L(A) \cap L(B)$]

Pf of claim: Induction on $|w|$.

Base case: $w = \varepsilon$.

$$\hat{\delta}((q,r),\varepsilon) \underset{\substack{\uparrow \\ \text{def of } \hat{\delta}}}{=} (q,r) = (\underset{\substack{\uparrow \\ \text{def of } \hat{\delta}_A}}{\hat{\delta}_A(q,\varepsilon)}, \underset{\substack{\uparrow \\ \text{def of } \hat{\delta}_B}}{\hat{\delta}_B(r,\varepsilon)})$$

$\therefore$ claim holds for $w = \varepsilon$. Base case ✓

Inductive case: $w \neq \varepsilon$, so $w = xa$ where
$x \in \Sigma^*$ is the principal prefix of w
$a \in \Sigma$ " " last symbol of w.

[$|x| < |w|$, so can assume the claim holds for x
"inductive hypothesis"]

$$\hat{\delta}((q,r),w) = \hat{\delta}((q,r),xa) \underset{\substack{\uparrow \\ \text{def of } \hat{\delta}}}{=} \delta(\underline{\hat{\delta}((q,r),x)}, a)$$

$$\underset{\substack{\uparrow \\ \text{ind. hyp.}}}{=} \delta((\underline{\hat{\delta}_A(q,x)}, \hat{\delta}_B(r,x)), a)$$

$$\underset{\substack{\uparrow \\ \text{def of } \delta}}{=} (\delta_A(\hat{\delta}_A(q,x),a), \delta_B(\hat{\delta}_B(r,x),a))$$

$$\underset{\substack{\uparrow \\ \text{defs of } \hat{\delta}_A \text{ and } \hat{\delta}_B}}{=} (\hat{\delta}_A(q,w), \hat{\delta}_B(r,w))$$ [$w = xa$]

☒ Claim.

Show that $L(A \wedge B) = L(A) \cap L(B)$: (5)

Let $w \in \Sigma^*$ be arbitrary.

$\underline{A \wedge B \text{ accepts } w} \underset{\text{def of acceptance in } A \wedge B}{\Longleftrightarrow} \hat{\delta}(\underbrace{(q_{0,A}, q_{0,B})}_{q_0}, w) \in \underbrace{F}_{F_A \times F_B}$

$\Updownarrow$

$w \in L(A \wedge B)$

$\underset{\text{by the claim}}{\Longleftrightarrow} \left(\hat{\delta}_A(q_{0,A}, w), \hat{\delta}_B(q_{0,B}, w)\right) \in F_A \times F_B$

$\underset{\text{by def of cartesian product}}{\Longleftrightarrow} \hat{\delta}_A(q_{0,A}, w) \in F_A \wedge \hat{\delta}_B(q_{0,B}, w) \in F_B$

$\Longleftrightarrow$ A accepts w <u>and</u> B accepts w

$\Longleftrightarrow w \in L(A) \cap L(B)$

$\therefore L(A \wedge B) = L(A) \cap L(B)$.

$\therefore$ Proposition //

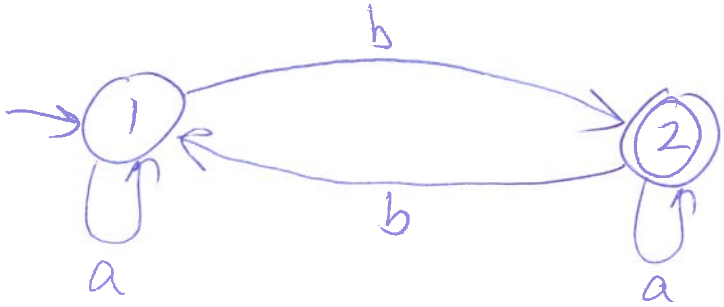
$A \wedge B$ ("product construction")

Example

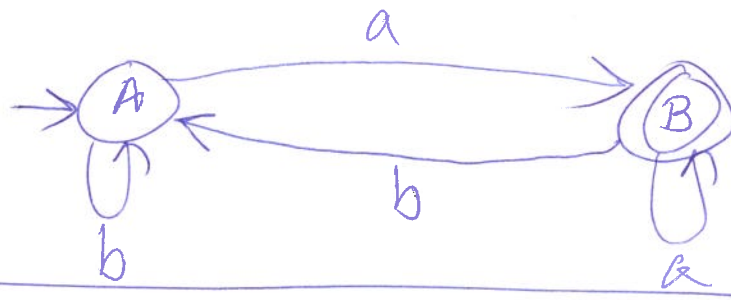
$\Sigma = \{a, b\}$

6

$A :=$

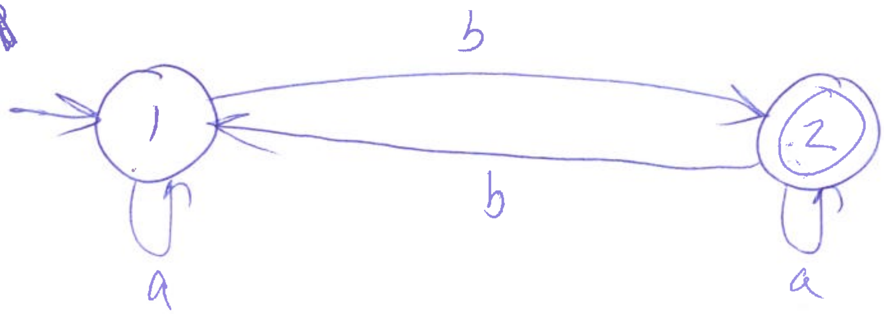


$B :=$

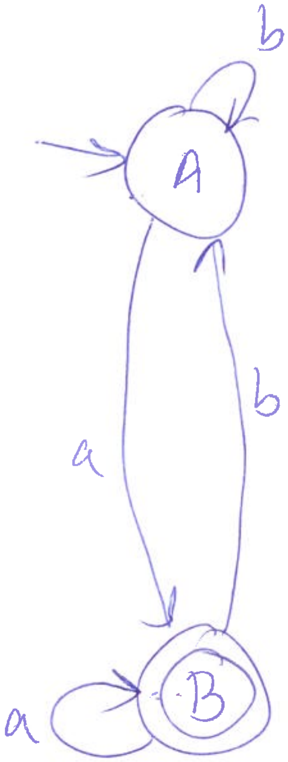


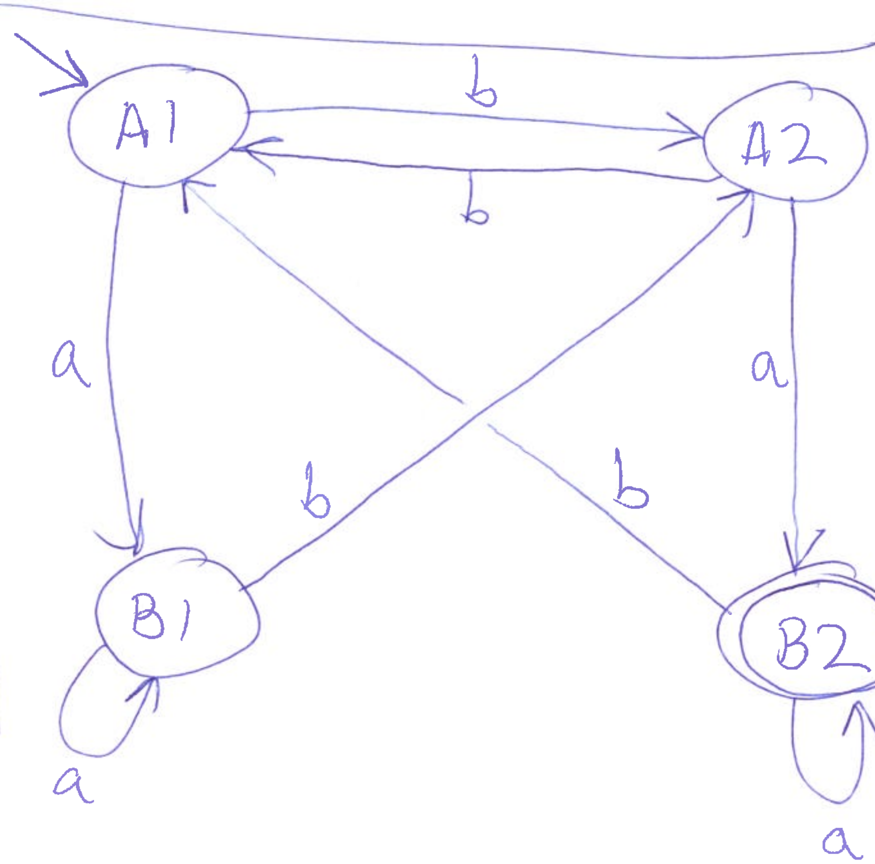
$B \wedge A$

A



B





Tabular Form of $B \cap A$

	a	b
$A\underline{1}$	B 1	A 2
A 2	B 2	A 1
$B\underline{1}$	B 1	A 2
B 2	B 2	$A\underline{1}$

<u>Corollary</u>: REG_{Σ} is closed under all Boolean operations: If L_1, L_2 are regular, then so are

$$\left.\begin{array}{l} \overline{L_1} \\ L_1 \cap L_2 \end{array}\right\} \text{already proved}$$

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

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

$$L_1 \Delta L_2 = (L_1 \setminus L_2) \cup (L_2 \setminus L_1)$$

$\vdots$

CSCE 355
1/23/2023

{1,2,3,4}

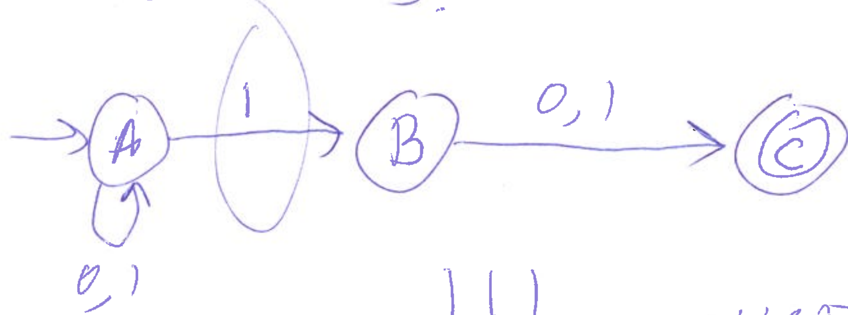
Nondeterministic Finite Automata (NFAs) ①

Relax the determinism restriction: any number of edges can leave the same state with the same label. NFA accepts a string w iff there is <u>some</u> choice of transitions that

1. end in an accepting state and
2. read the whole string

$L_1 = \{ w \in \{0,1\}^* : \text{The 2nd last symbol of } w \text{ is a } 1 \}$

A:

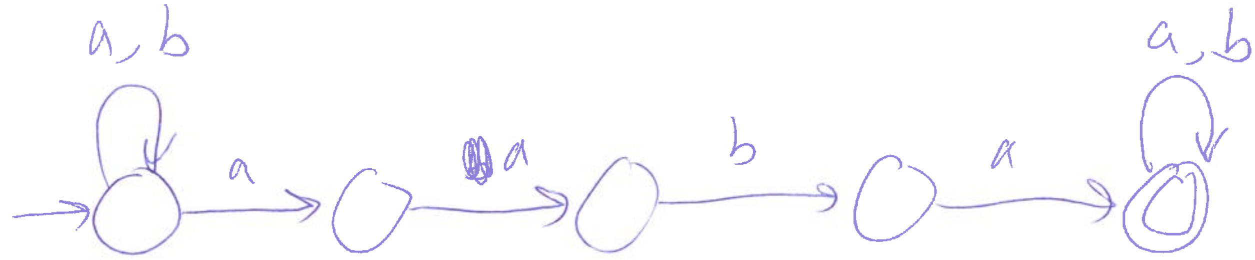


recognizes L

111 AABC	accept
101	reject

$L_2 = \{ w \in \{a,b\}^* : w \text{ contains aaba as a substring} \}$

B:



<u>Def:</u> A <u>nondeterministic finite automaton</u> (<u>NFA</u>) is a 5-tuple $\langle Q, \Sigma, \delta, q_0, F \rangle$ where Q, Σ, q_0, F are as with a DFA and

$$\delta : Q \times \Sigma \longrightarrow \underbrace{2^Q}_{\text{also called } \mathcal{P}(Q)\text{, the powerset of } Q\text{, i.e., the set of all subsets of } Q}$$

So $\underbrace{\delta(q, a)}_{\text{states reachable from } q \text{ by following an edge labeled with } \underline{a}} \subseteq Q \quad \forall q \in Q, \forall a \in \Sigma$

A in tabular form:

	0	1
→ A	{A}	{A, B}
B	{C}	{C}
* C	∅	∅

Def: Let $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ be an NFA and $w \in \Sigma^*$. A complete computation path of A on input w is a sequence of states $s_0, s_1, \ldots, s_n \in Q$ such that there exist symbols $w_1, \ldots, w_n \in \Sigma$ such that

1. $w = w_1 \cdots w_n$
2. $s_0 = q_0$
3. For every $1 \leq i \leq n$,

$$s_i \underbrace{\in}_{\text{"member of"}} \underbrace{\delta(s_{i-1}, w_i)}_{\text{set of states}}$$

Say that the path ends in s_n.

Path is accepting if it ends in an accepting state ($s_n \in F$) otherwise rejecting.

A accepts w if there exists an accepting path of A on w.

L(A), the lang recognized by A is the same as with DFAs.

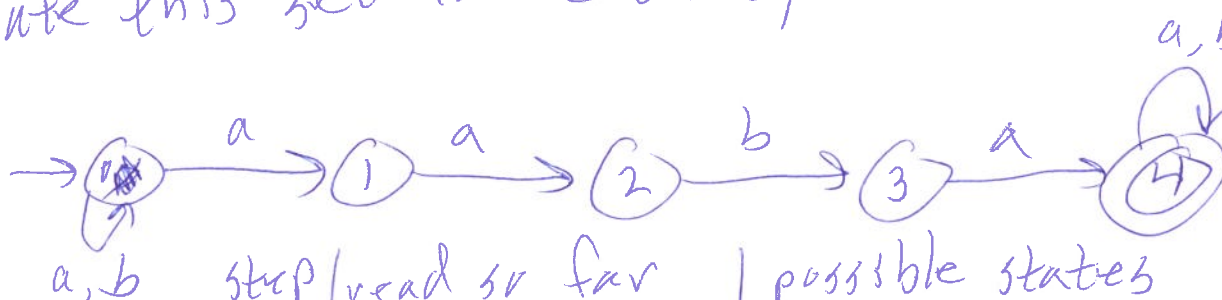
Note: A DFA can be trivially converted into an equivalent NFA.
recognizing the same language

<u>Theorem</u>: For every NFA there exists an equivalent DFA.

How to simulate an NFA efficiently.

On input w, read w symbol by symbol, keeping track of the set of states possibly reachable having read so far. Update this set for each symbol read.

<u>Ex</u>:



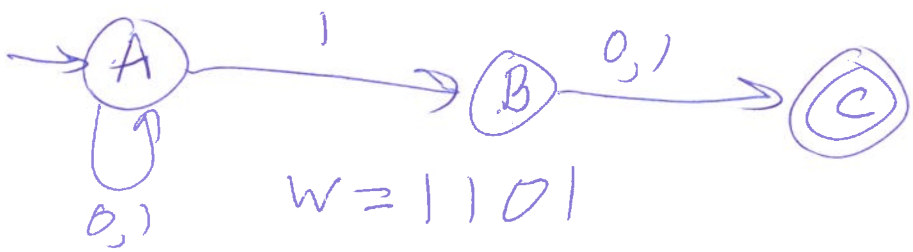
w = baaabaa

step	read so far	possible states
0	ε	0
1	b	0
2	ba	0,1
3	baa	0,1,2
4	baaa	0,1,2

(5)

5	baaab	0,3
6	baaaba	0,1,4
7	baaabaa	0,1,2,4

Ex:



$w = 1101$

Step		
0	ε	A
1	1	AB
2	11	ABC
3	110	AC
4	1101	AB

reject

"Proof" of the theorem: Idea: states of the DFA are sets of states of the NFA.

Given NFA $A := \langle Q, \Sigma, \delta, q_0, F \rangle$, define DFA

$$D := \langle 2^Q, \Sigma, \Delta, Q_0, \mathcal{F} \rangle$$

where

$$Q_0 := \{q_0\},$$

$\mathcal{F} := \{ S \subseteq Q : S \cap F \neq \emptyset \}$ ⑥

and for any $S \subseteq Q$ and $a \in \Sigma$,

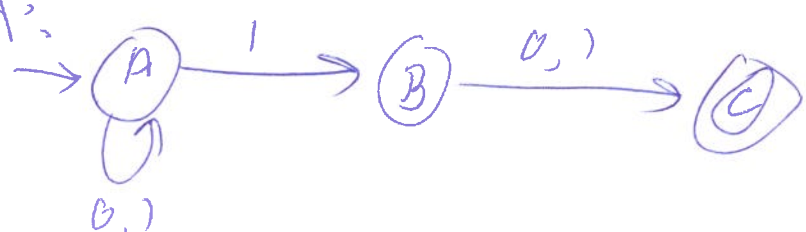
$$\Delta(S, a) := \bigcup_{q \in S} \delta(q, a)$$

$$= \{ r \in Q : \exists q \in S,\ r \in \delta(q, a) \}$$

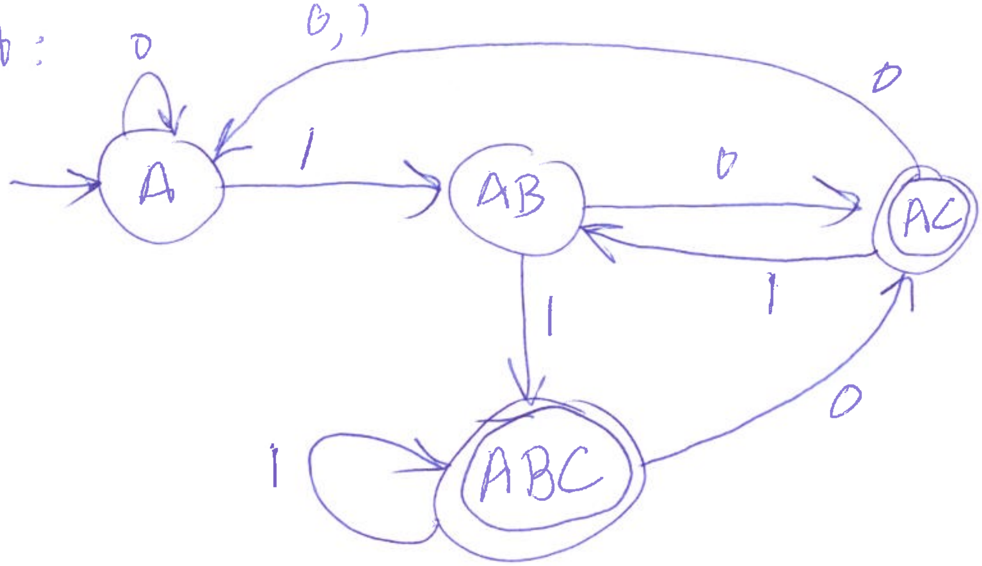
~~Proof~~ Then $L(D) = L(A)$.

Proof of correctness omitted. //

Ex: NFA:



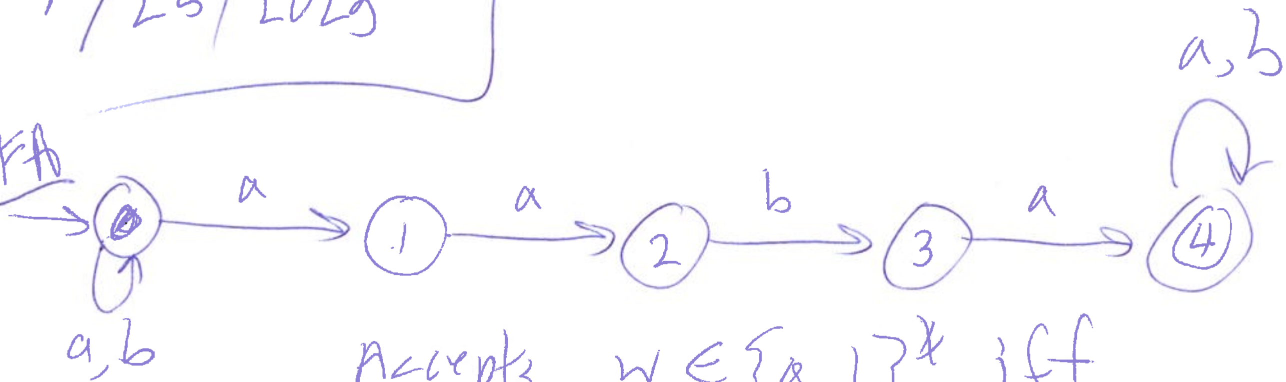
DFA:



CSCE 355
1/25/2023

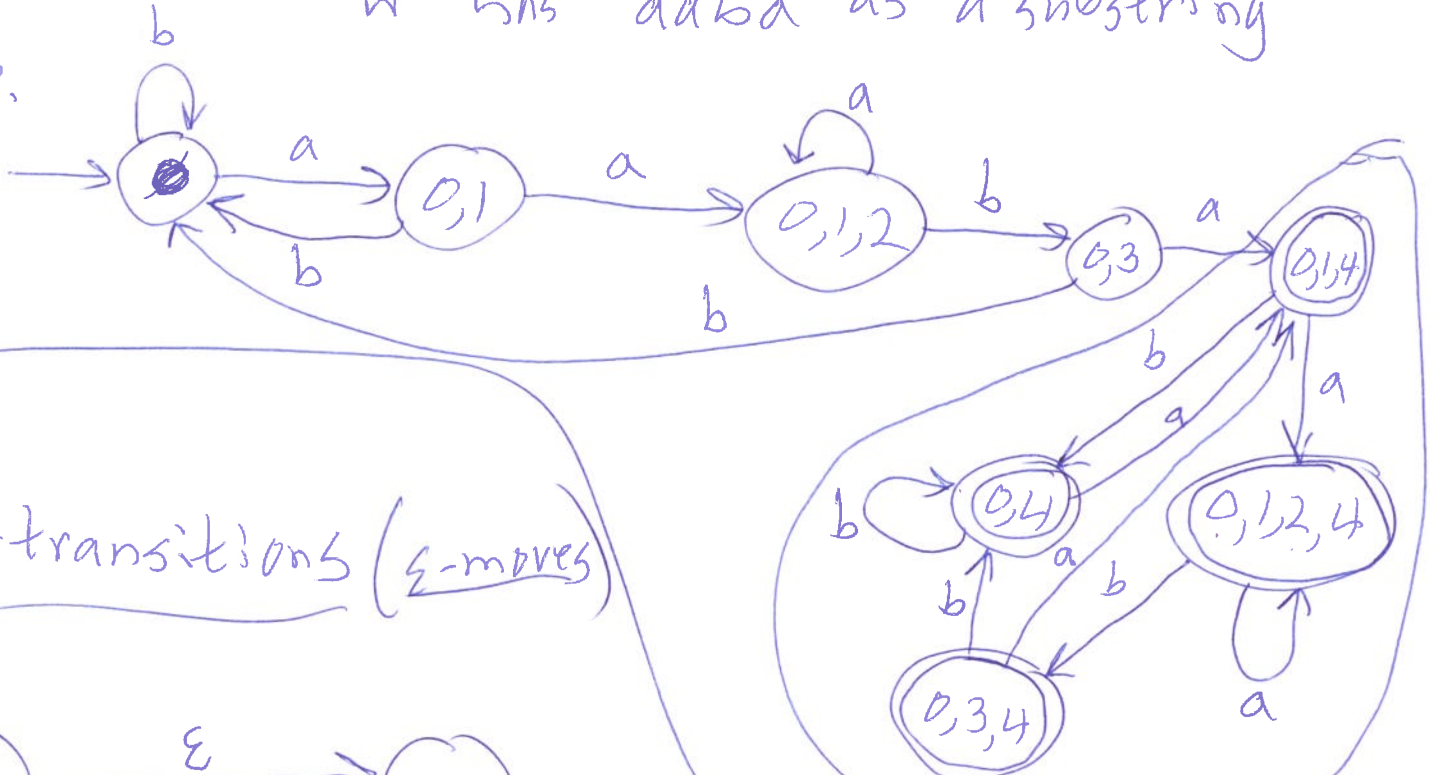
NFA → DFA example

NFA



Accepts $w \in \{a,b\}^*$ iff
w has aaba as a substring

DFA:



ε-transitions (ε-moves)

means $\underline{\delta(q,\varepsilon)}$ contains r

An ε-NFA is an NFA that allows <u>ε-moves</u>.

<u>Def</u>: An <u>ε-NFA</u> is a 5-tuple $\langle Q, \Sigma, \delta, q_0, F \rangle$ where Q, Σ, q_0, F are as with an NFA, and

$$\delta : Q \times (\underbrace{\Sigma \cup \{\varepsilon\}}_{\substack{\text{strings of} \\ \text{length 0 or 1}}}) \longrightarrow \cancel{Q}\ 2^Q$$

Def: A (complete) comp. path of an ε-NFA $N = \langle Q, \Sigma, \delta, q_0, F \rangle$ on input $w \in \Sigma^*$ is a sequence of states $s_0, s_1, \ldots, s_k \in Q$ ~~where each $s_i \in \Sigma \cup \{\varepsilon\}$~~ such that there exist $w_1, w_2, \ldots, w_k \in \Sigma \cup \{\varepsilon\}$ such that

1. $w = w_1 \cdots w_k$ (now $k \geq |w|$)
2. $s_0 = q_0$
3. For all i, $1 \leq i \leq k$

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

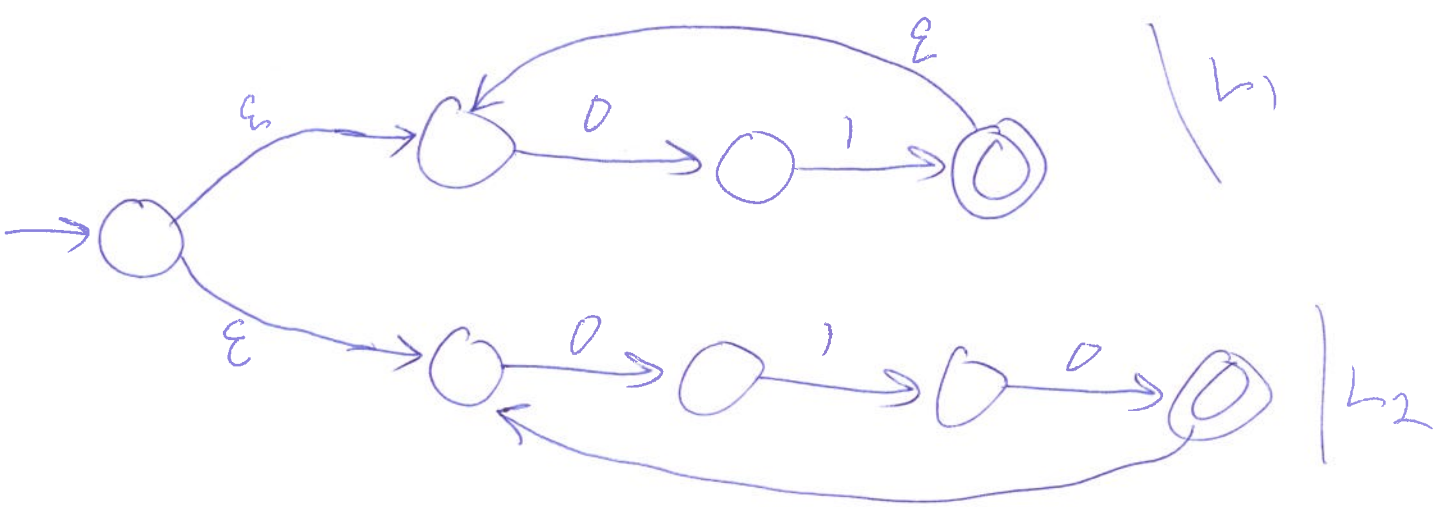
Say the path ends in s_k. N accepts w means there exists a complete comp. path on w ending in some accepting state ($s_k \in F$).

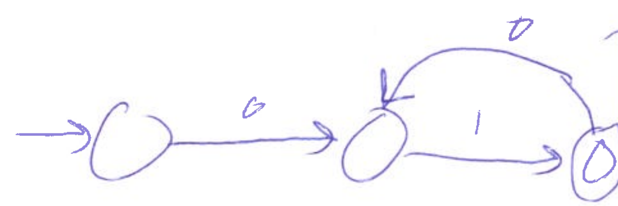
Ex: $L = \{ w \in \{0,1\}^* :$ w is either
1 or more reps of 01
or 1 or more reps of 010 $\}$

$L = L_1 \cup L_2$ where

$L_1 = \{ \ldots$ 1 or more reps of 01 $\}$

$L_2 = \{ \ldots$ 1 or more reps of 010 $\}$



[DFA for L_1: ]

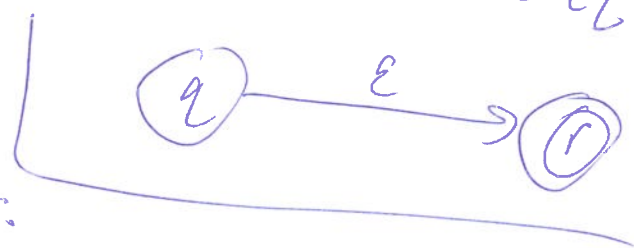
ε-moves can be removed entirely, giving an equivalent NFA with no more states than the original ε-NFA.

Theorem: For every ε-NFA there exists an equivalent NFA with the same state set.

Idea: add new non-ε transitions to bypass all ε-moves, which then are redundant and can be removed.

Proof: Let $N = \langle Q, \Sigma, \delta, q_0, F \rangle$ be any ε-NFA.

Step 1: ~~while~~ while there exist states $q, r \in Q$ such that $q \notin F$ and $r \in F$ and $r \in \delta(q, \varepsilon)$ do



make q accepting:

$$F := F \cup \{q\}$$

[bypasses ε-moves at the end of an accepting path]

step 2: // bypass ε-moves followed by non-ε-moves

while there exist $q, r, s \in Q$ ~~such~~ and $a \in \Sigma$ such that

not necessarily distinct

$r \in \delta(q, \varepsilon)$ and $s \in \delta(r, a)$ and $s \notin \delta(q, a)$

do: Add s to $\delta(q, a)$: $\delta(q, a) := \delta(q, a) \cup \{s\}$

Step 3: (only do after finishing steps 1 & 2):

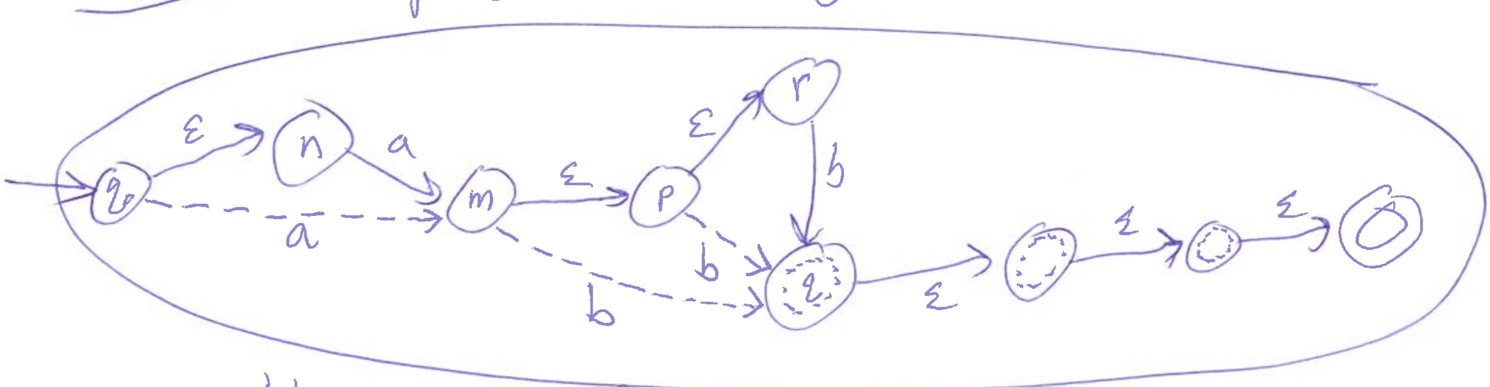
Remove all ε-moves from N:

$$\forall q \in Q, \quad \delta(q, \varepsilon) := \emptyset.$$

Observe: steps 1 & 2 don't cause any string to be rejected that was accepted by the original N. (~~as~~ only change states from rejecting to accepting and only add new transitions). Thus any accepting path in ~~the~~ N before ~~after~~ steps 1 & 2 is an accepting path on the same string after steps 1 & 2.

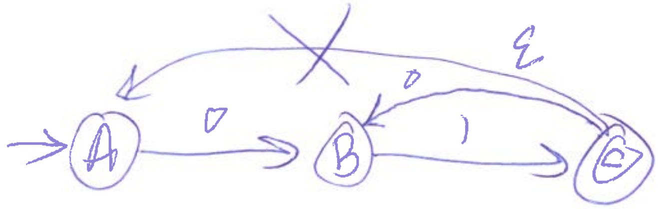
Claim: Any string accepted by the original N is accepted by the N after step 3.

Idea (example): In original N



accepting path of string ab

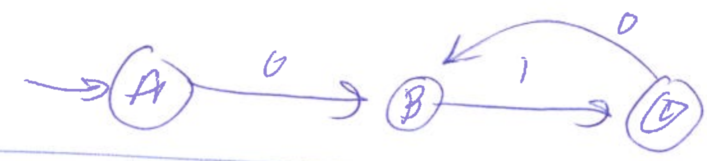
Ex:

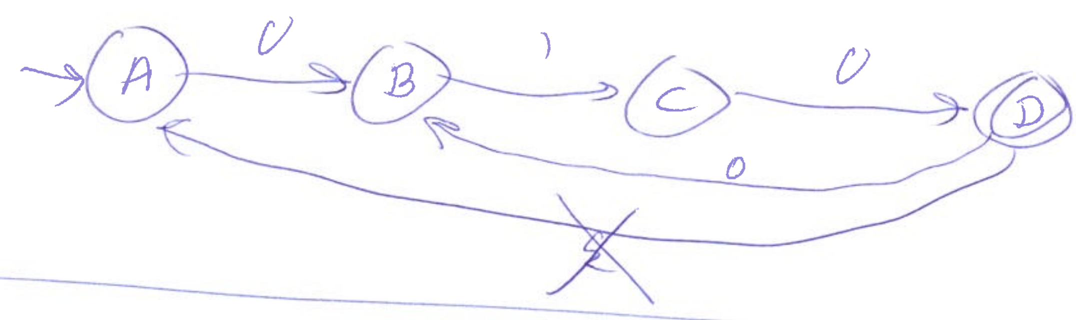
N = 

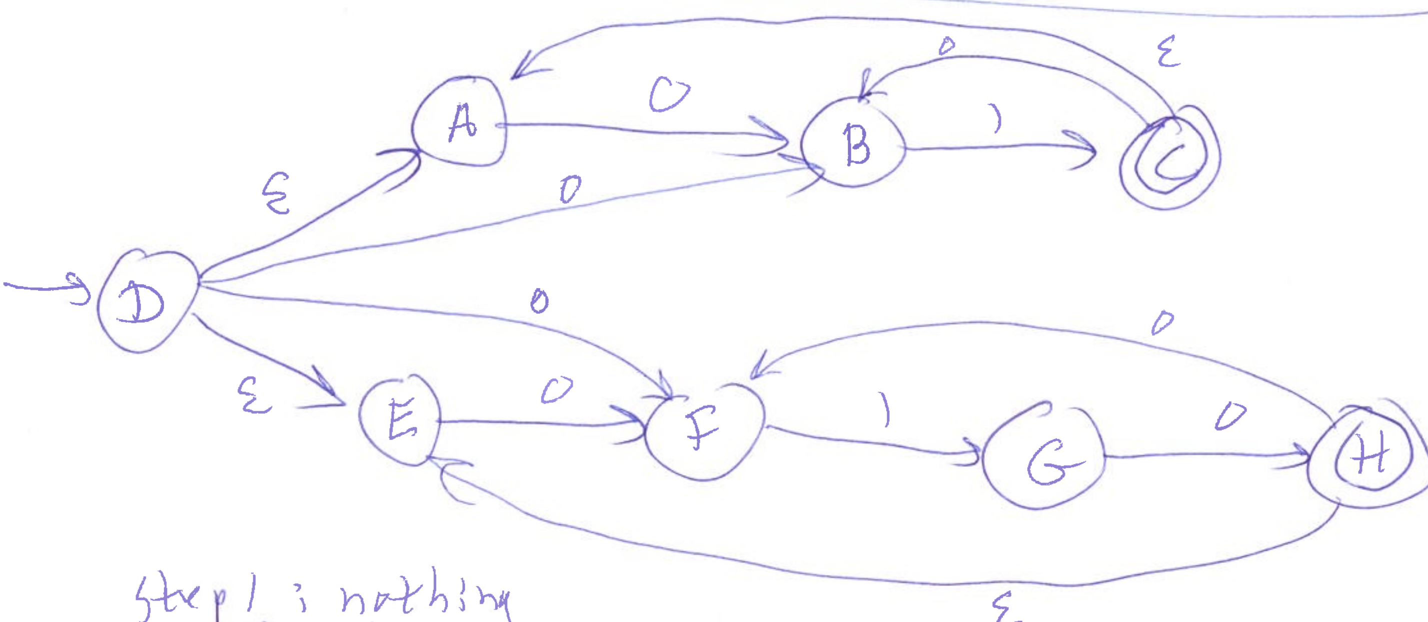
step 1: nothing to do.

step 2: add 0-transition from C to B
nothing more to do.

step 3: remove ε-move from C to A



N = 

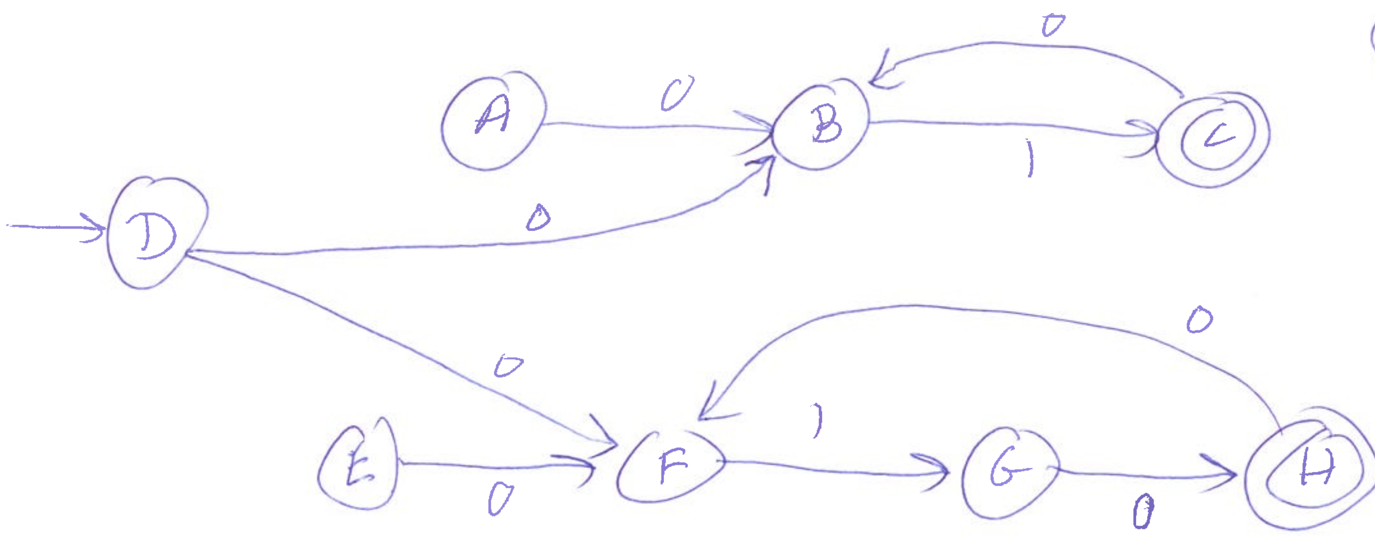


step 1: nothing

step 2: add $C \xrightarrow{0} B$, $H \xrightarrow{0} F$, $D \xrightarrow{0} F$, $D \xrightarrow{0} B$

step 3:

7



Can remove A & E (unreachable from D)

CSCE 355
1/30/2023

Two topics:

1. DFA minimization algo
2. Intro to regular expressions

DFA minimization

(A) A DFA is sane if every state is reachable from the start state:

$A = \langle Q, \Sigma, \delta, q_0, F \rangle$ is sane iff $\forall q \in Q, \exists w \in \Sigma^*, \hat{\delta}(q_0, w) = q$

A DFA that is not sane is not minimal.

1st step in DFA minimization: remove all state unreachable from the start state; result is an equivalent, sane DFA.

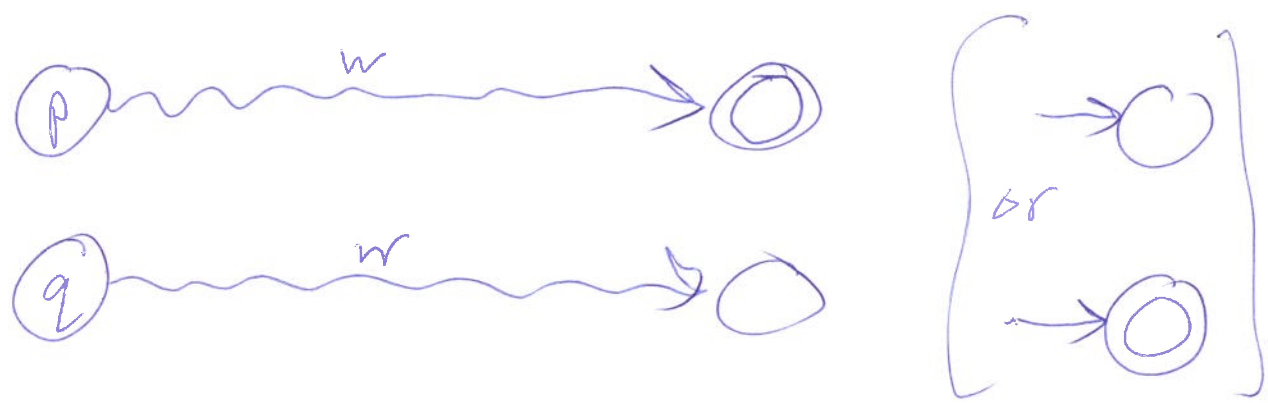
(B) Assume our input DFA is sane.
We find groups of mutually indistinguishable (indist.) states, merge each group into a single state.

Def: Let $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ be a DFA and let $p, q \in Q$ be states of A.

Say that p & q are distinguishable (dist.) ②
if there exists some string $w \in \Sigma^*$ such that
one of $\hat{\delta}(p,w)$ and $\hat{\delta}(q,w)$ is accepting
and the other rejecting. In this case,
say that w distinguishes p from q.

[Future accepting behavior of A ~~depends on~~
differs between states p & q if w is
~~left~~ remaining on the input.]

p & q are indist. if they are not
dist., i.e., if no string distinguishes them.



Setup for algo to find all distinguishable pairs of states:

Maintain a table $T[\cdot,\cdot]$ 2-dim, each dim indexed by states

Initially, $T[p,q]$ is blank for all $p,q \in Q$.

// when we find a pair of dist. states,
// we mark the T-entry with an 'x'.

// T is symmetric: T[p,q] == T[q,p]

// T[q,q] stays blank throughout

Step 1 (base case): For every p, q such that one is accepting & the other rejecting,

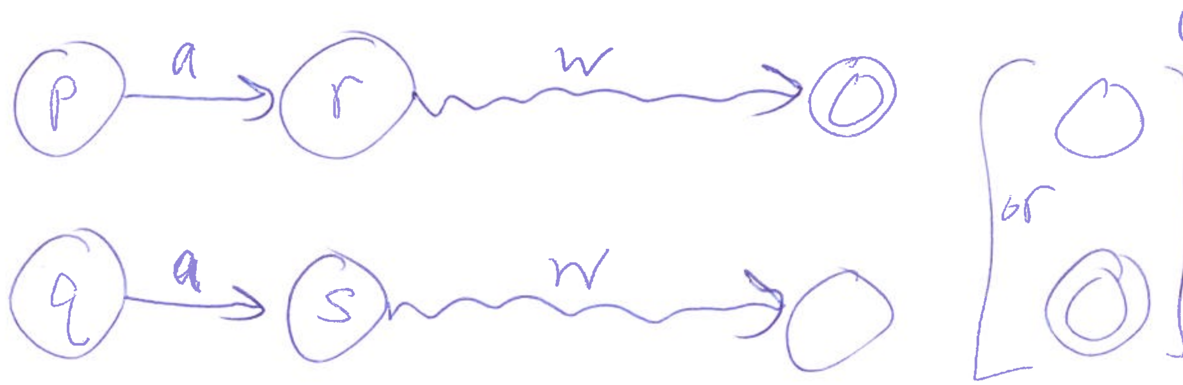
$$T[p,q] := T[q,p] := \text{'x'}$$

{p, q are dist. by ε}

Step 2: (iterative case)

while (there exist states $p, q \in Q$ and $a \in \Sigma$ such that $T[p,q]$ is blank but $T[\delta(p,a), \delta(q,a)] == \text{'x'}$), do

$$T[p,q] := T[q,p] := \text{'x'}$$



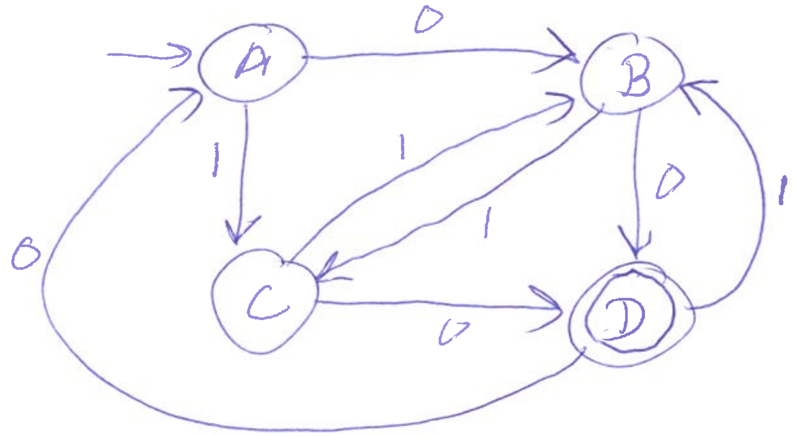
suppose $T[r,s] == $ 'X'

then $\exists w$, $\hat{\delta}(r,w)$ is acc
$\hat{\delta}(s,w)$ is ~~a~~ rej | or vice versa

But then aw distinguishes p from q,
so safe to set $T[p,q]$ to 'X'

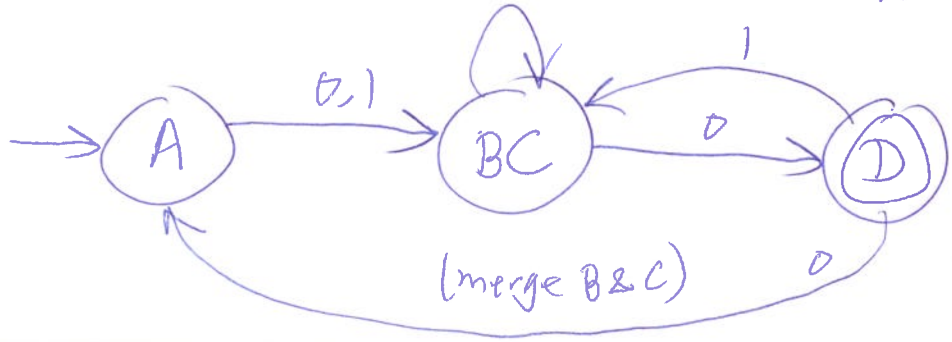
That's the whole algo to find <u>all</u> dist. pairs.
All dist pairs are found in steps (1) or (2)

<u>Example:</u>

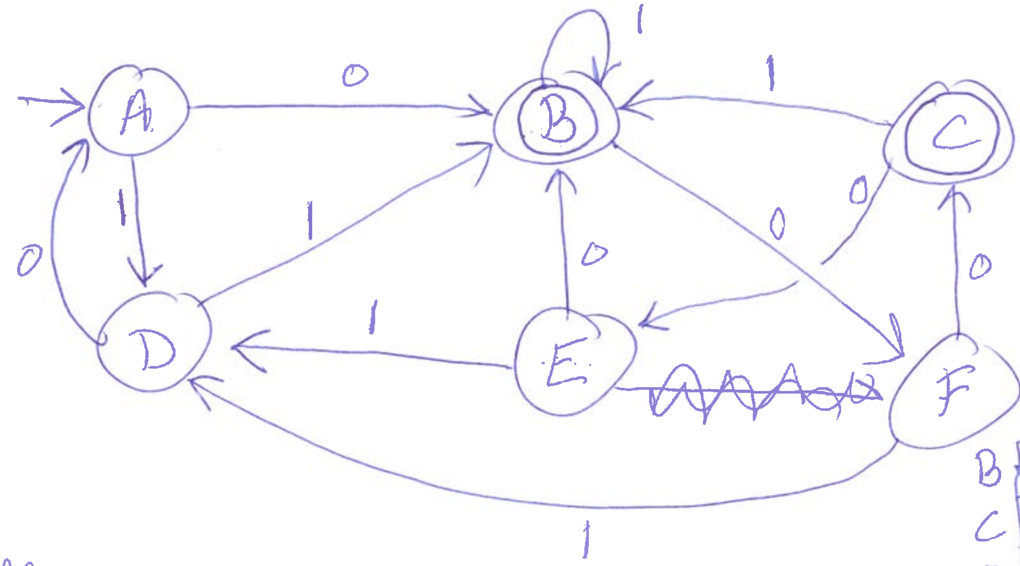


	A	B	C
B	X		
C	X	O	
D	X	X	X

Min equiv DFA:



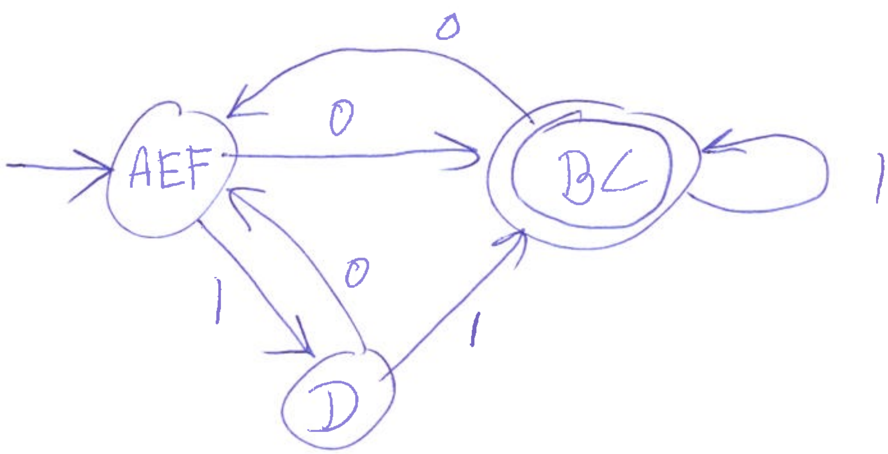
Ex:



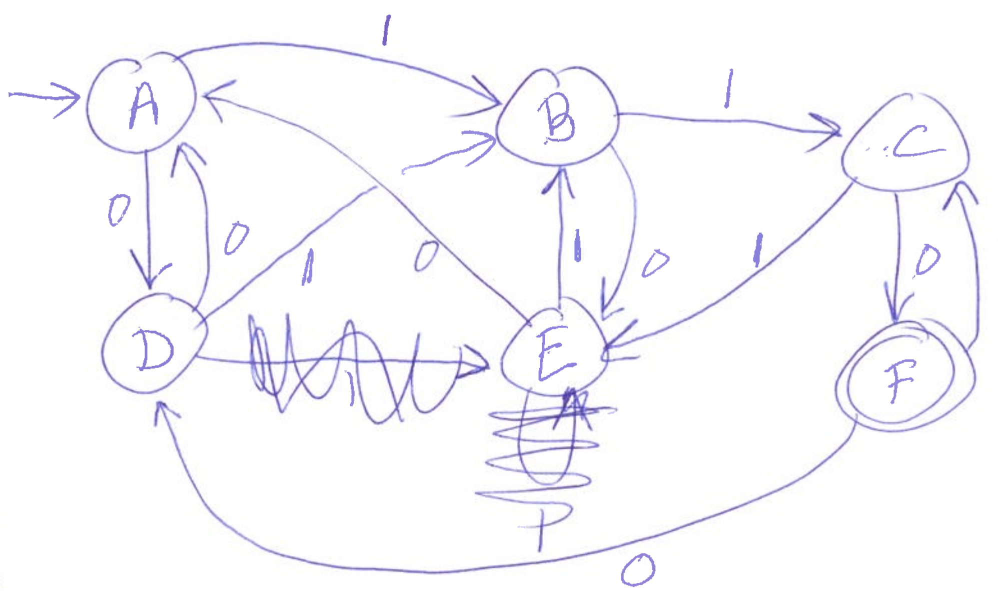
B	X				
C	X	O			
D	X	X	X		
E	O	X	X	X	
F	O	X	X	X	O
	A	B	C	D	E

Merge: B & C, AEF

Min DFA



Ex:



B	X				
C	X	X			
D		X	X		
E		X	X		
F	X	X	X	X	X
	A	B	C	D	E

Done (?)