# Computation Classes: An Overview

- Finite State Machines
  (Regular Languages)

- Pushdown Automata
  (Context-Free Languages)

- Decidable Problems
  (Turing Machines that Halt)

- Undecidable Problems
  Turing Recognizable
  (Turing Machines that may not halt.)

**Non-Determinism**

Will add power in the Context-Free Group, but not elsewhere.

# CHAPTER 1:
# REGULAR LANGUAGES

FINITE STATE MACHINE F.S.M.

ALSO: "FINITE AUTOMATON"

The simplest model of computation.

Small Computer or Controller

Limited Memory

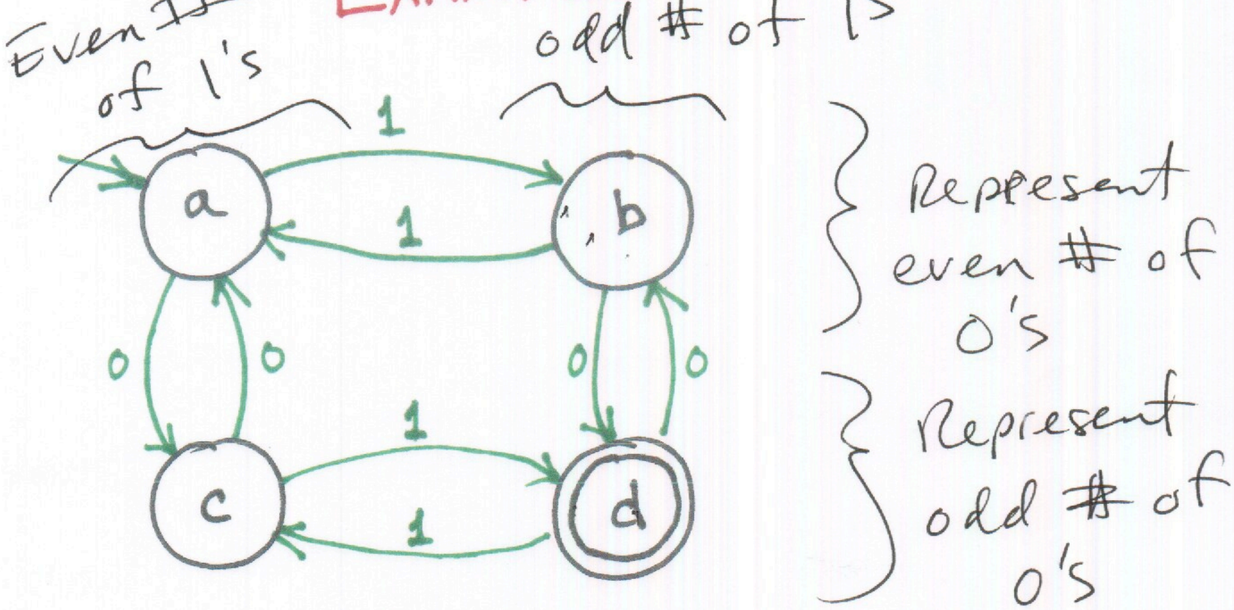
Finite & usually quite small.

REGULAR LANGUAGES

REGULAR EXPRESSIONS

2

# FINITE STATE MACHINE EXAMPLE




STATES (NODES)

TRANSITIONS (EDGES)

STARTING STATE

Always exactly one starting state

the "initial state"

ACCEPTING STATE (or "final" states)

May be more than 1

ALPHABET OF SYMBOLS

$\Sigma = \{0, 1\}$

## CAN GENERATE STRINGS

- START AT STARTING STATE.
- TAKE TRANSITIONS AT RANDOM.
- FINISH UP ONLY IN AN "ACCEPTING" STATE.
- THE SET OF STRINGS YOU CAN GENERATE?

## CAN RECOGNIZE STRINGS

- START IN STARTING STATE.
- START AT FIRST SYMBOL IN THE STRING.
- FOLLOW TRANSITIONS AS DETERMINED BY THE SYMBOLS IN THE STRING.
- PROCESS <u>ALL</u> SYMBOLS IN STRING
- DO YOU END UP IN AN "ACCEPTING" STATE OR NOT?
- THE SET OF STRINGS THAT ARE ACCEPTED?
- OTHERS ARE "REJECTED."

4

# FORMAL DEFINITION OF FINITE STATE MACHINE

Described by a 5-tuple:

$$M = (Q, \Sigma, \delta, q_0, F)$$

$Q$ = Set of States $\{a, b, c, d\}$

*Finite* Number of states

$\Sigma$ = Alphabet, a *Finite* Set of Symbols

$\delta$ = The TRANSITION FUNCTION

$$\delta : Q \times \Sigma \rightarrow Q$$

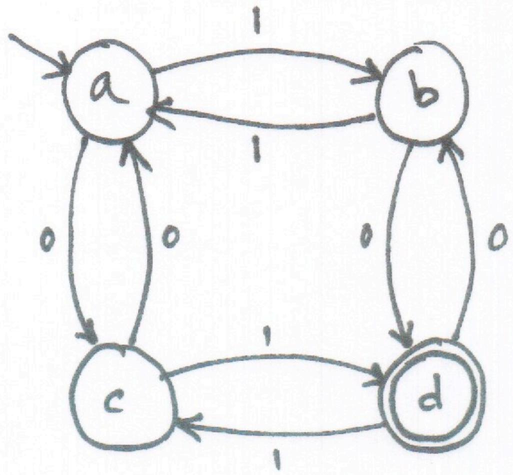
$q_0$ = The STARTING STATE

$q_0 \in Q$ (or "INITIAL" STATE)

$F$ = The set of ACCEPT states (or "FINAL" STATES)

$$F \subseteq Q$$

5




$Q = \{ a, b, c, d \}$

$\Sigma = \{ 0, 1 \}$

$q_0 = a$

$F = \{ d \}$

$\delta =$

Symbols

| States | 0 | 1 |
|---|---|---|
| a | c | b |
| b | d | a |
| c | a | d |
| d | b | c |

"THE LANGUAGE THAT $M$ ACCEPTS IS $A$."

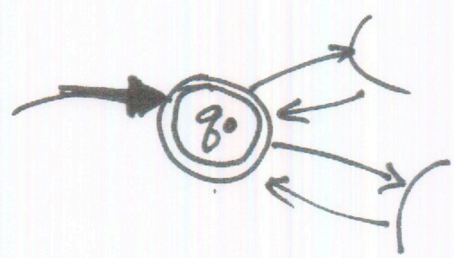
"THE LANGUAGE OF $M$"

"$M$ RECOGNIZES $A$." ← yes

"$M$ ACCEPTS $A$." $M$ accepts/rejects Strings

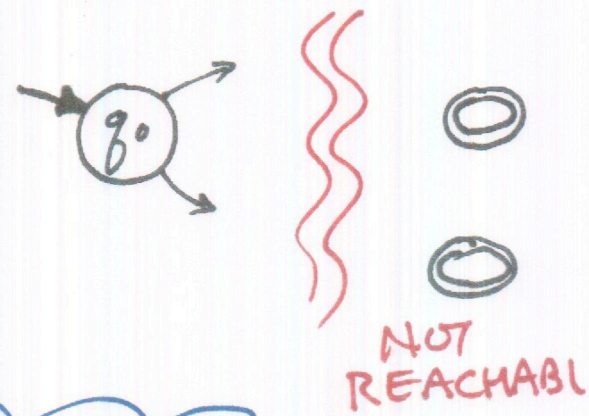
THE EMPTY STRING

$\epsilon$ (epsilon, $\varepsilon$)



THE EMPTY LANGUAGE

$\emptyset = \{\}$




NOTE:

$\{\epsilon\} \neq \emptyset$

$\epsilon \neq \emptyset$

If a machine accepts NO strings
then it recognizes
the EMPTY LANGUAGE

7

# DESIGN EXAMPLE

$\Sigma = \{0, 1\}$

Want to recognize...

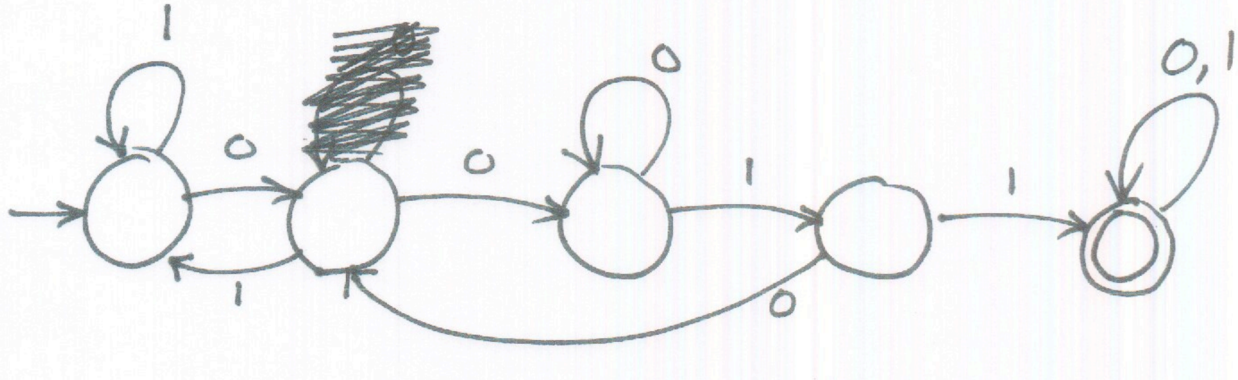
Any string that does NOT contain 0011 in it.

How to proceed?

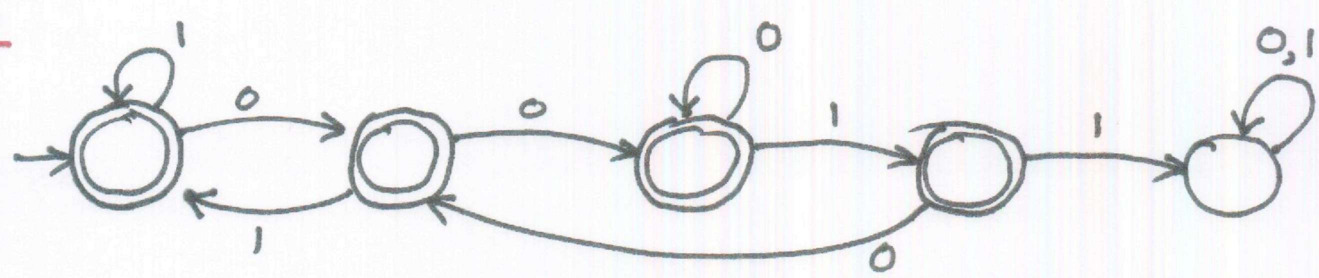
Try a simpler problem.

A string that does contain 0011 in it.

$M_1$



$M_2$



## Terminology
A FSM "accepts" a string.
A FSM "recognizes" a language.

## Notation
$L(M_1)$ = The language that $M_1$ recognizes.
= The set of strings over $\{0,1\}^*$ that contain 0011 as a substring.

$L(M_2)$ = The set of strings over $\{0,1\}^*$ that do not contain 0011.

## COMPLIMENTING A LANGUAGE

They are sets, after all.

$$L(M_1) = \overline{L(M_2)}$$

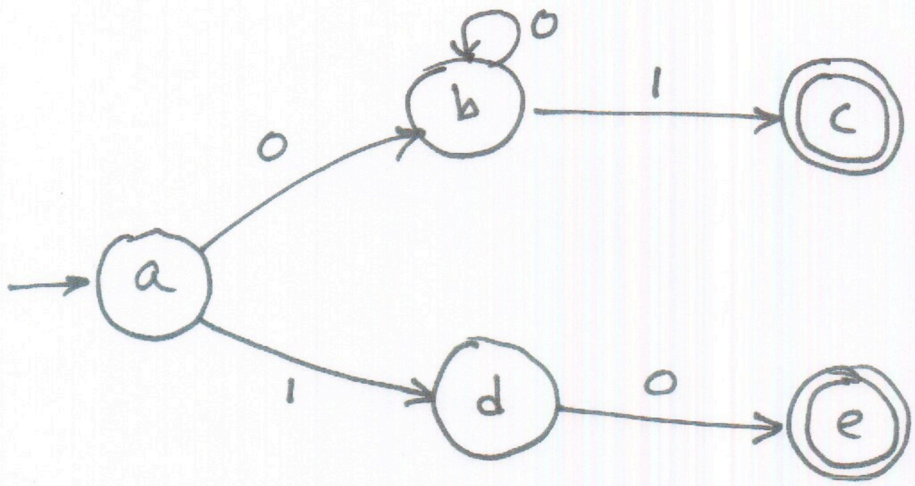
The "Universe"
All possible strings made with symbols from the alphabet.

$\Sigma = \{0,1\}$ Universe = $\{0,1\}^*$

Set compliment is always relative to some Universe; (implicitly).

9

# WHAT DOES THIS F.S.M. RECOGNIZE?



Recognizes 10

Also 01, 001, 0001, ... $0^+1$

L = { w | w is either 10 <u>or</u> a string of at least one 0 followed by a single 1 }

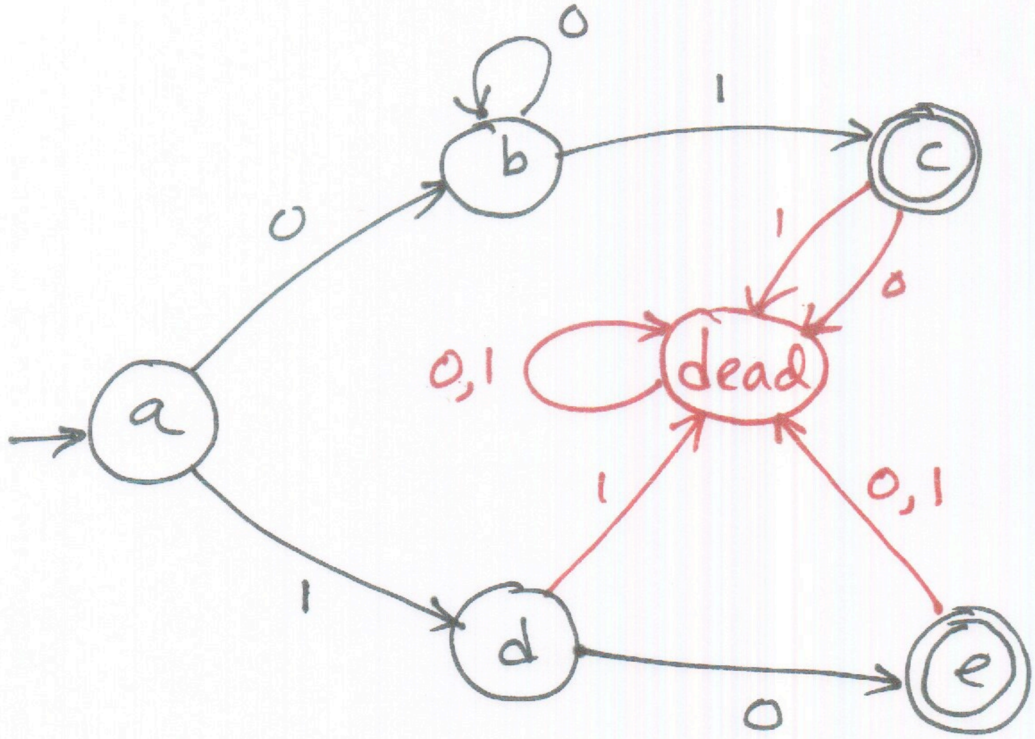
What about

111
1010

} What happens?

10

# DEAD STATES



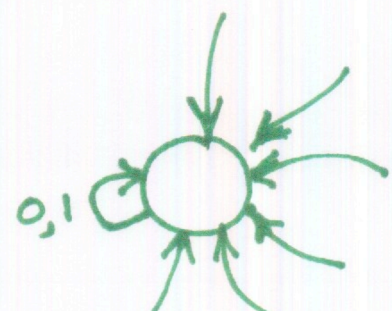

$\delta$ is a function

FORMALLY, must be defined

$$\delta(c, 1) = ?$$

If some transitions are missing, add a dead state.

(Often, prefer not to show dead state.)



# FORMAL DEFINITION OF COMPUTATION

Let $M = (Q, \Sigma, \delta, q_0, F)$

Let $w = w_1 w_2 \ldots w_N$ be a string
where $w_i \in \Sigma$

M accepts $w$ if there is a sequence of states
$r_0, r_1, r_2, \ldots, r_N$ in $Q$
such that

$r_0 = q_0$

$\delta(r_i, w_{i+1}) = r_{i+1}$ for $0 \le i < N$

$r_N \in F$

**We say...**

M "recognizes" Language A
if $A = \{ w \mid M \text{ accepts } w \}$

## DEFINITION

A language is a **REGULAR LANGUAGE** iff some Finite State Machine recognizes it.

What languages are NOT regular?

Anything that requires memory.

The F.S.M. memory is <u>very</u> limited

Cannot store the string.

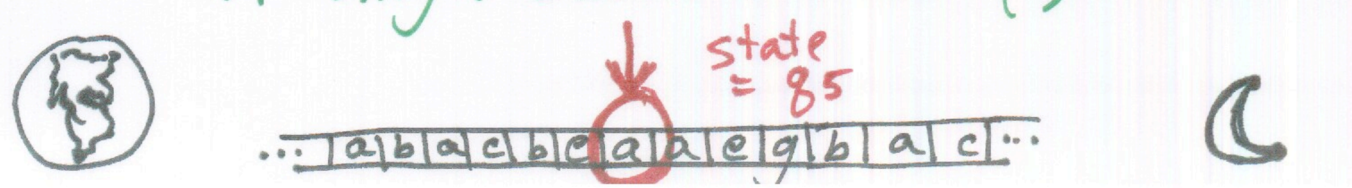
Cannot "count."

Not Regular:

$ww$ $\qquad \underbrace{01101}_{w}\underbrace{01101}_{w}$

$0^N 1^N$ $\qquad \underbrace{000000}_{6}\underbrace{111111}_{6}$

Imagine a String from here to the Moon. You are trying to recognize it. Your only memory: A single small number (i, # of states)




13

# DESIGNING A FSM

Binary Numbers that are divisible by 3.

$\Sigma = \{0, 1\}$ $\quad L = \{ \underset{0}{0}, \underset{3}{11}, \underset{6}{110}, \underset{9}{1001}, \underset{12}{1100}, \underset{15}{1111} \ldots \}$

---

As we scan a binary number, what does each bit do to the value?

$$10110101011110001 \begin{cases} 0 \longrightarrow 2(x) \\ 1 \longrightarrow 2(x)+1 \end{cases}$$

3X ← Divisible by 3

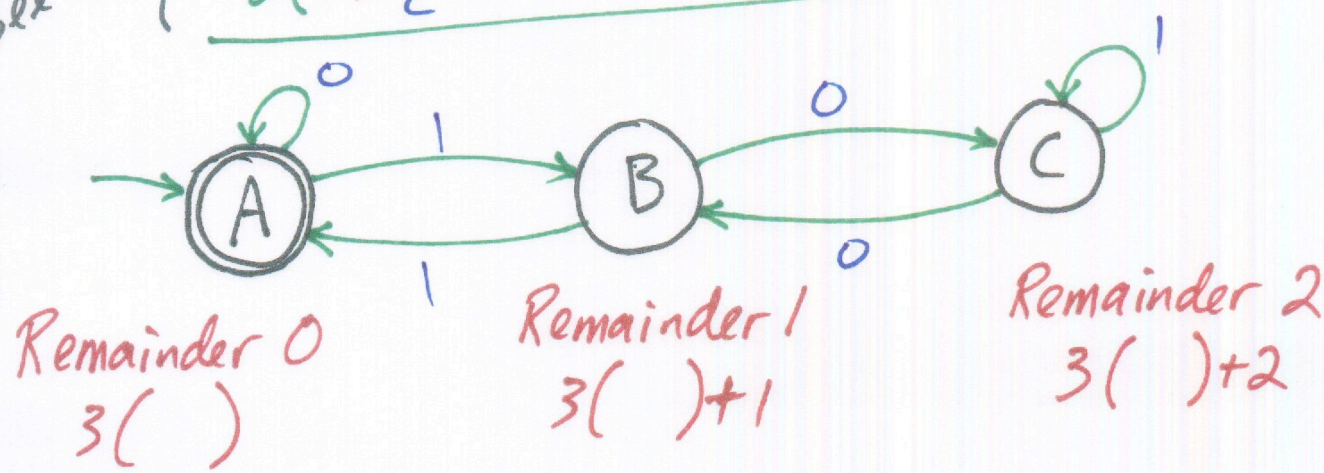
3X+1, 3X+2 ↔ Not divisible by 3

if we see 0:

| | | | |
|---|---|---|---|
| $2(3X)^{A}$ | $=$ | $3(2X)$ | A |
| $2(3X+1)^{B}$ | $=$ | $3(2X)+2$ | C |
| $2(3X+2)^{C}$ | $=$ | $3(2X+1)+1$ | B |

if we see 1:

| | | | |
|---|---|---|---|
| $2(3X^{A})+1$ | $=$ | $3(2X)+1$ | B |
| $2(3X+1^{B})+1$ | $=$ | $3(2X+1)$ | A |
| $2(3X+2^{C})+1$ | $=$ | $3(2X+1)+2$ | C |




14

# REGULAR OPERATIONS ON LANGUAGES

UNION

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$$

CONCATENATION

$$A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$$

STAR "closure"

$$A^* = \{x_1 x_2 \cdots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$$

EXAMPLE

$$\Sigma = \{a, b, c, \ldots z\}$$
$$A = \{aa, b\}$$
$$B = \{x, yy\}$$

$$A \cup B = \{aa, b, x, yy\}$$
$$A \circ B = \{aax, aayy, bx, byy\}$$
$$A^* = \{\epsilon, aa, b, aaaa, aab, baa, bb, aaaaaa, aaaab, aabaa, aabb, \ldots$$

15

## THEOREM

The class of Regular Languages is CLOSED under UNION.

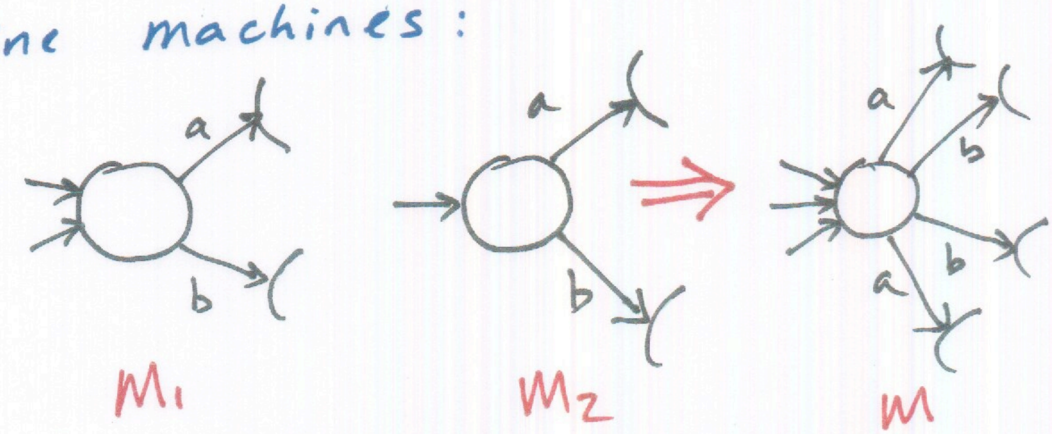
If $L_1$ and $L_2$ are regular languages, then so is $L_1 \cup L_2$.

## PROOF (BY CONSTRUCTION)

Assume $L_1 = L(M_1)$
$L_2 = L(M_2)$

Build $M$ to recognize $L_1 \cup L_2$.

Combine machines:




WHOOPS!
Not a F.S.M!

16

What about running $M_1$ then trying $M_2$?

NO! CAN'T REWIND THE INPUT!

IDEA: Simulate $M_1$ and $M_2$ simultaneously.

Each state in $M$ corresponds to *two* states.

$$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$$
$$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$$

Construct

$$M = (Q, \Sigma, \delta, q_0, F)$$

Assume alphabets are the same.

( OR: $\Sigma = \Sigma_1 \cup \Sigma_2$ )

APPROACH:

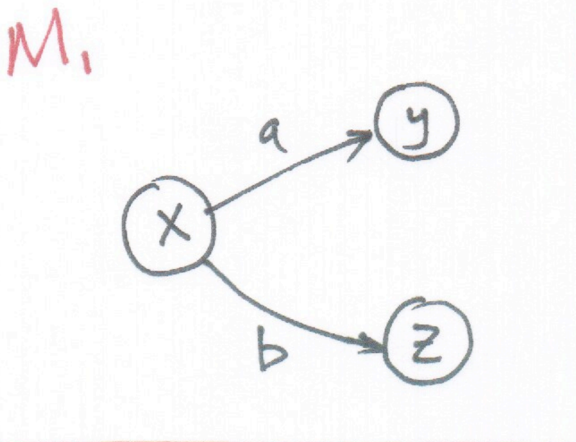
Each state in new machine represents *two* states

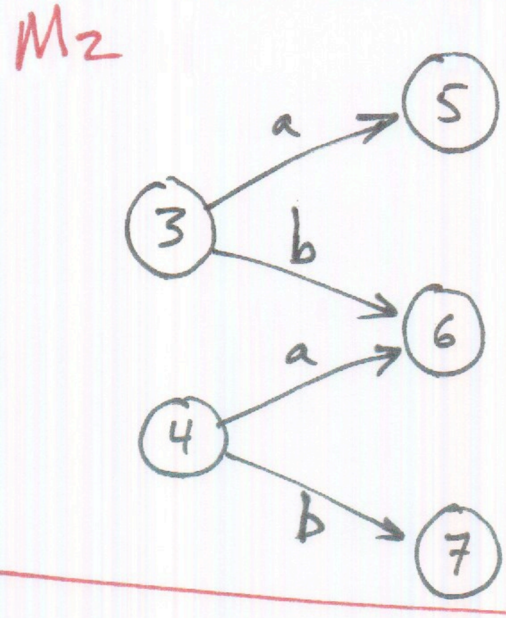
One from $M_1$

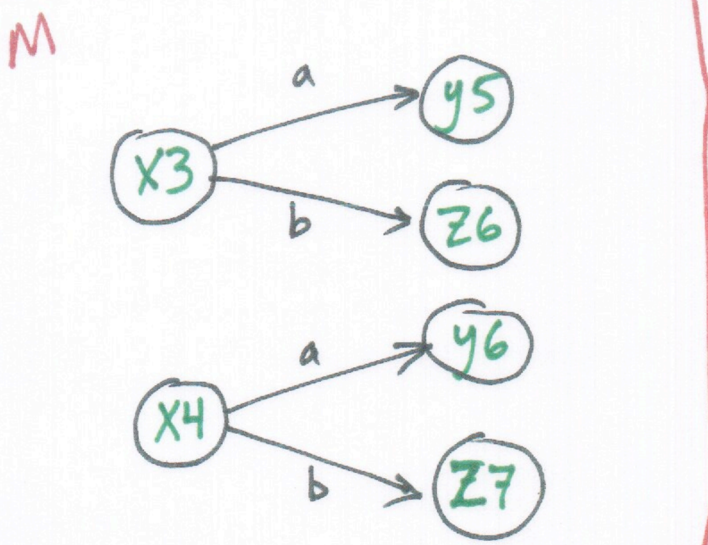
One from $M_2$

(Lots of possible combinations.)

17




$$Q = Q_1 \times Q_2 = \{ (r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2 \}$$

$$\delta((r_1, r_2), a) = (\delta_1(r_1, a),\ \delta_2(r_2, a))$$

$$q_0 = (q_1, q_2)$$

$$F = \{ (r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2 \}$$

Accept states in M?

If either $M_1$ or $M_2$ would be in an accept state.

## THEOREM

THE CLASS OF REGULAR LANGUAGES IS CLOSED UNDER CONCATENATION.

If $L_1$ and $L_2$ are regular then so is $L_1 \circ L_2$.

## PROOF

CAN'T DO IT YET.

WE NEED...

**NONDETERMINISM**

19

# DETERMINISM

"GIVEN THE CURRENT STATE, WE KNOW WHAT THE NEXT STATE WILL BE."

ONLY ONE UNIQUE NEXT STATE.

NO CHOICES.

NO RANDOMNESS. (Perfect REPEATABILITY)

NO ORACLES.

NO CHEATING. (No errors or malfunctions)

NORMAL COMPUTERS ARE DETERMINISTIC

Except they take random inputs.
(e.g. the timing of keystrokes)

Random errors may preclude repeatability.

SO FORGET ABOUT IT!

20

# NONDETERMINISM

"GIVEN THE CURRENT STATE, THERE MAY BE MULTIPLE NEXT STATES."

- The next state is chosen at random.
- All next states are chosen in parallel and pursued simultaneously.

---

Same Thing

FSM means Deterministic Finite State Automaton/Machine

DFA means DETERMINISTIC FINITE STATE AUTOMATON

---

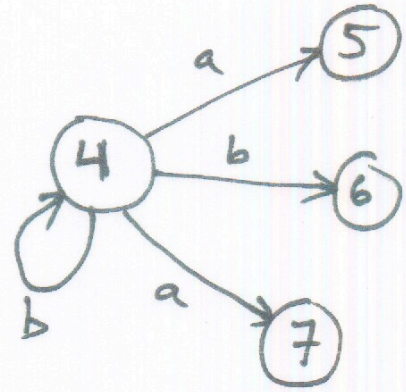
NFA means NONDETERMINISTIC FINITE STATE AUTOMATON

---

20.1

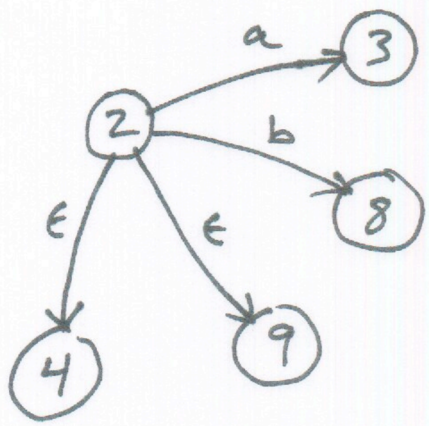
NOW WE WILL ALLOW

- MULTIPLE EDGES WITH THE SAME LABEL OUT OF A NODE.




Which edge should you take???

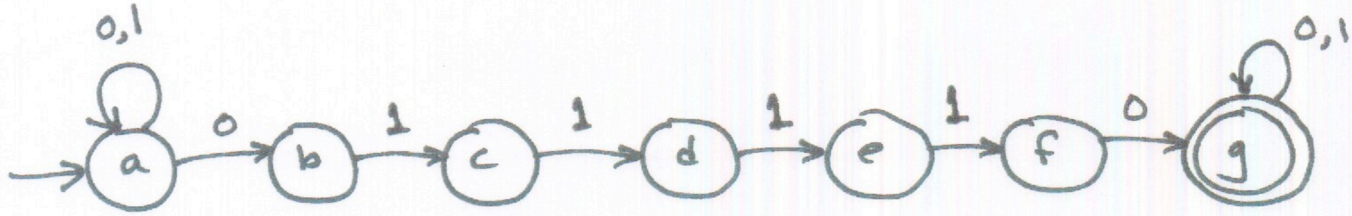
- EPSILON EDGES




Can take an ε-edge without scanning a symbol.
It is "OPTIONAL"!

## EXAMPLE

All strings that contain 011110




EXAMPLE STRING: 0100011110101

Lots of bad choices
that don't work,
that don't reach an accept state

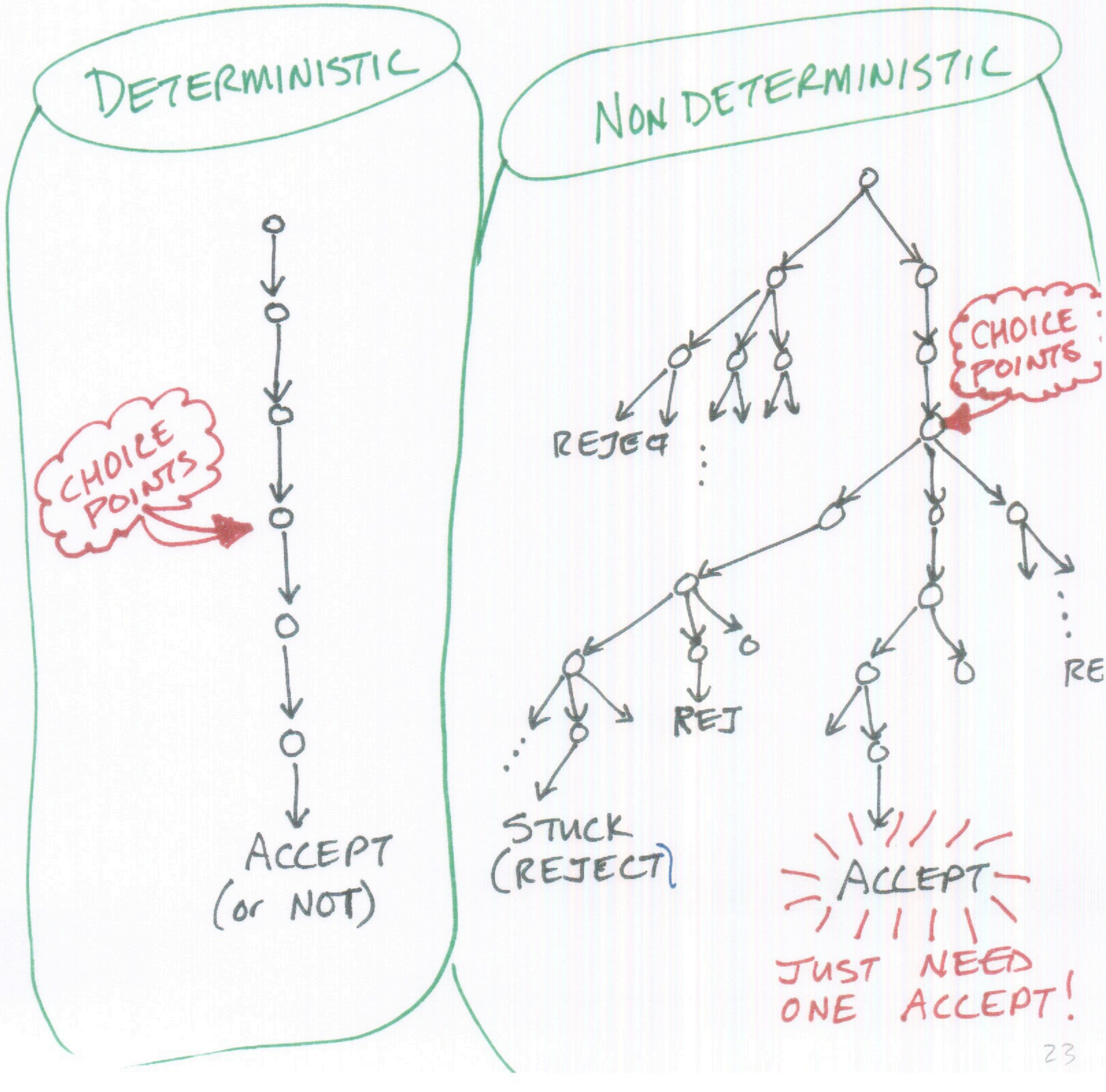
All we need is one way
to reach ACCEPT.

> If there is any way to run
> the machine that ends
> with ACCEPT,
> Then the NFA accepts.

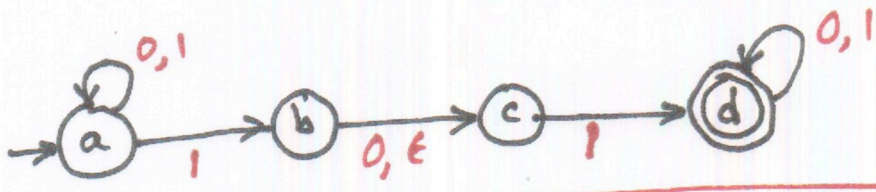
LOTS OF CHOICES - WHICH ONE TO TRY?

* Try them all.
* Make the right choice at each point.

$\left.\right\}$ EQUIVALENT

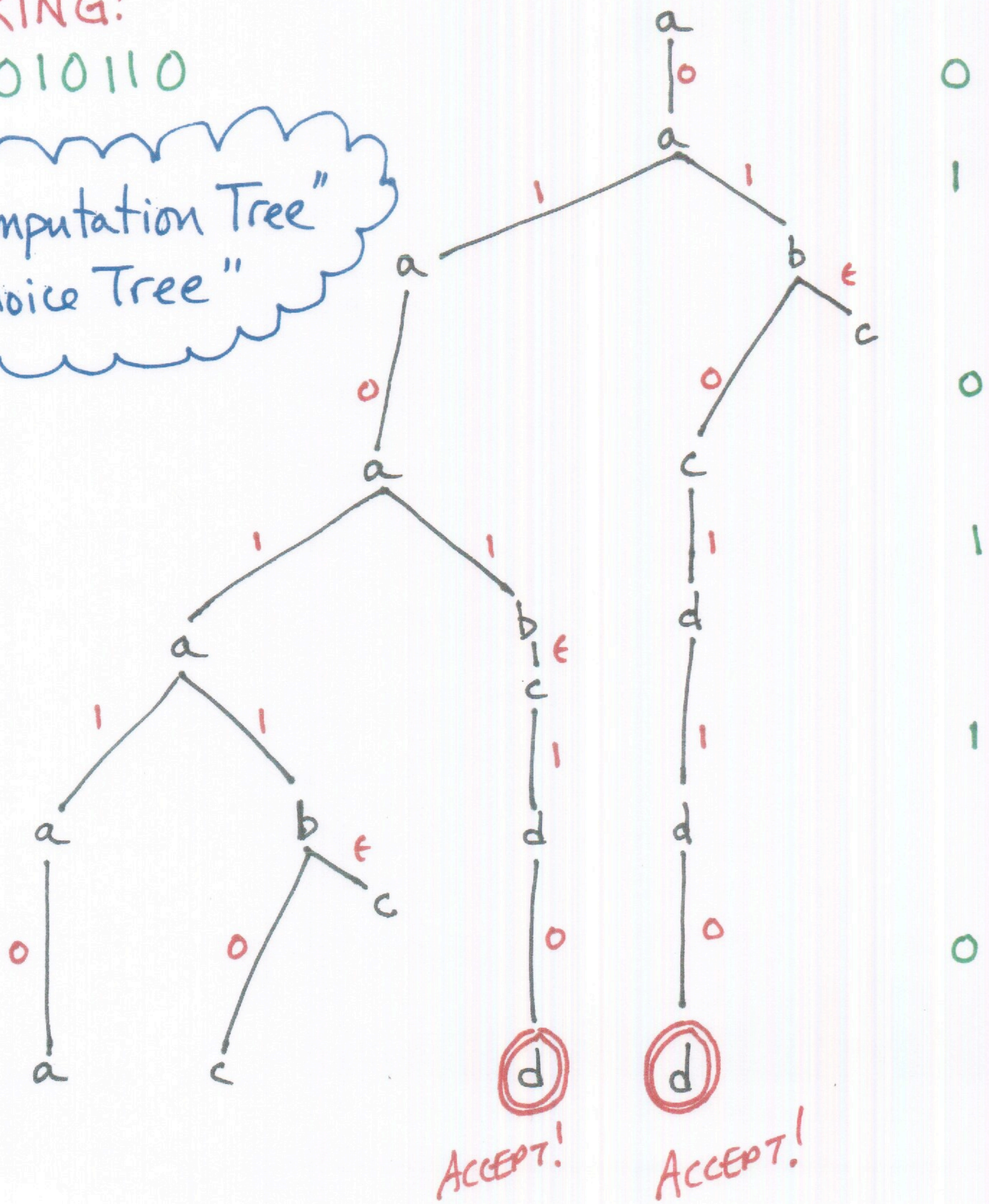


## EXAMPLE




STRING:
010110

"Computation Tree"
"Choice Tree"



## THEOREM

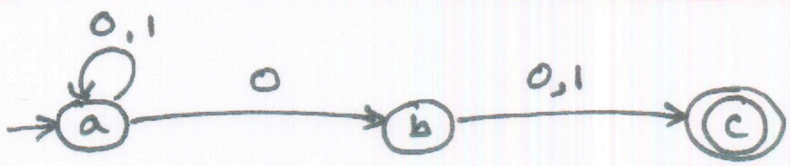
FOR EVERY NONDETERMINISTIC F.S.M.
THERE IS AN EQUIVALENT DETERMINISTIC F.S.M.

... But it may be large and hard to find!

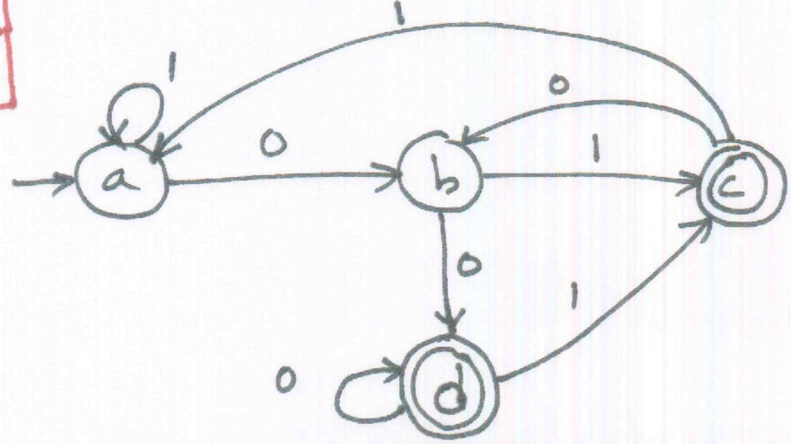
### EXAMPLE

All strings over $\{0,1\}^*$ that have a "0" in the second to the last position.

**NFA**




**FSM**
**DFA**



## EXAMPLE

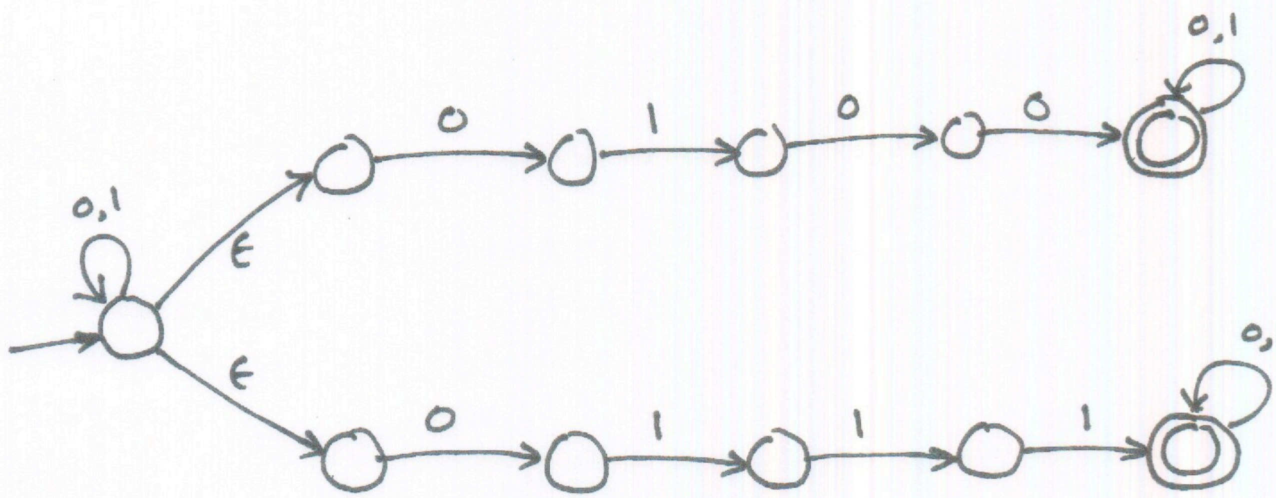
String contains either

... 0100 ...

or ... 0111 ...

When to start looking?
Which string to look for?

NONDETERMINISM!




## CHALLENGE:

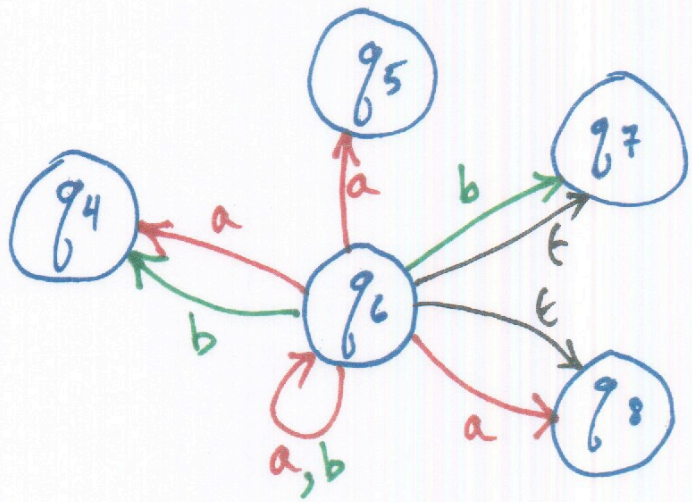
Build/Design a DFA to recognize this language.

26

**POWERSET** The set of all subsets.

$P(\{a,b,c\})$

$\emptyset \quad \{a\} \quad \{b\} \quad \{c\} \quad \{a,b\} \quad \{a,c\} \quad \{b,c\} \quad \{a,b,c\}$




If you are in state $q_6$:

... And you see an "a" $\{q_4, q_5, q_6, q_8\}$

... And you see a "b" $\{q_4, q_6, q_7\}$

... And you see "ε" $\{q_7, q_8\}$

27

# FORMAL DEFINITION OF NONDETERMINISTIC FINITE STATE MACHINE

$\Sigma$ = Alphabet

$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$

$M = (Q, \Sigma, \delta, q_0, F)$

$Q$ = States

member of, not Epsilon.

$\Sigma$ = Alphabet

$q_0$ = Start State, $q_0 \in Q$

$F$ = Accept States, $F \subseteq Q$

$\delta$ = Transition Function

$$\delta : Q \times \Sigma_\epsilon \rightarrow P(Q)$$

27.1

$\delta : Q \times \Sigma_\epsilon \rightarrow P(Q)$

BEFORE:

symbols

| States | a | b | c |
|---|---|---|---|
| $q_0$ | $q_1$ | $q_2$ | $q_4$ |
| $q_1$ | $q_2$ | $q_0$ | $q_1$ |
| ⋮ | ⋮ | ⋮ | ⋮ |

FOR NFA:

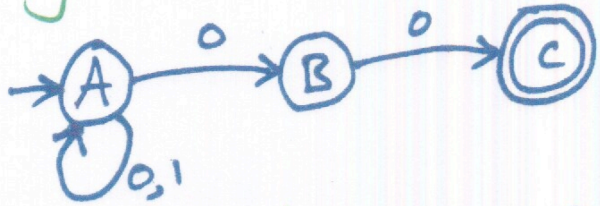
symbols

| States | a | b | c | $\epsilon$ |
|---|---|---|---|---|
| $q_0$ | $\{q_1, q_2\}$ | $\emptyset$ | $\{q_4, q_2\}$ | $\{q_0\}$ |
| $q_1$ | $\{q_4, q_2, q_1\}$ | $\{q_5, q_0\}$ | $\emptyset$ | $\{q_4, q_3\}$ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

27.2

EXAMPLE

Accept all strings over $\{0,1\}^*$ ending with "00."



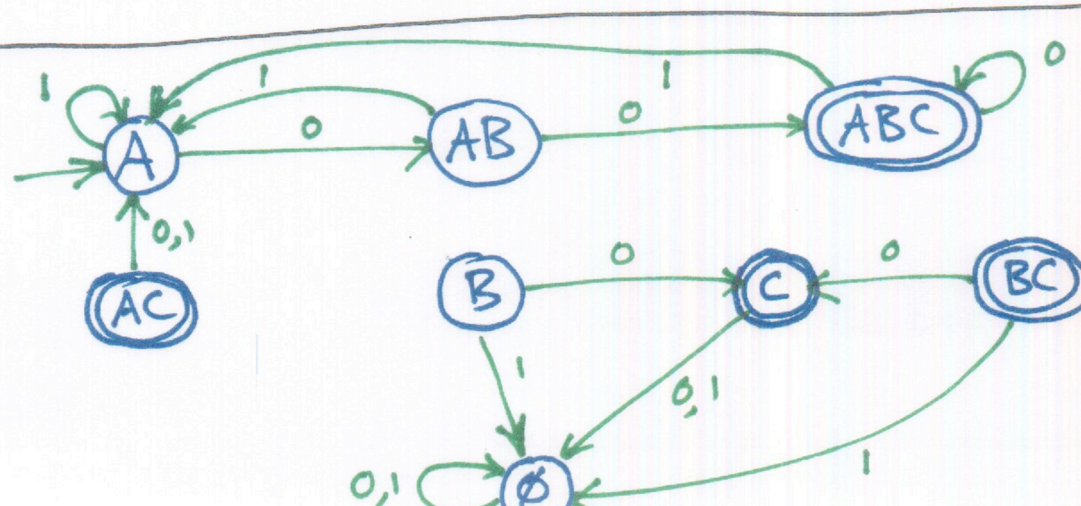

String to check: 00100

Simulate the execution. Put a finger on any state we could be in.

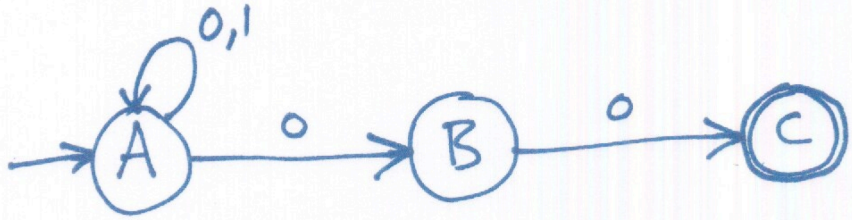
$\emptyset$ A B C AB BC AC ABC

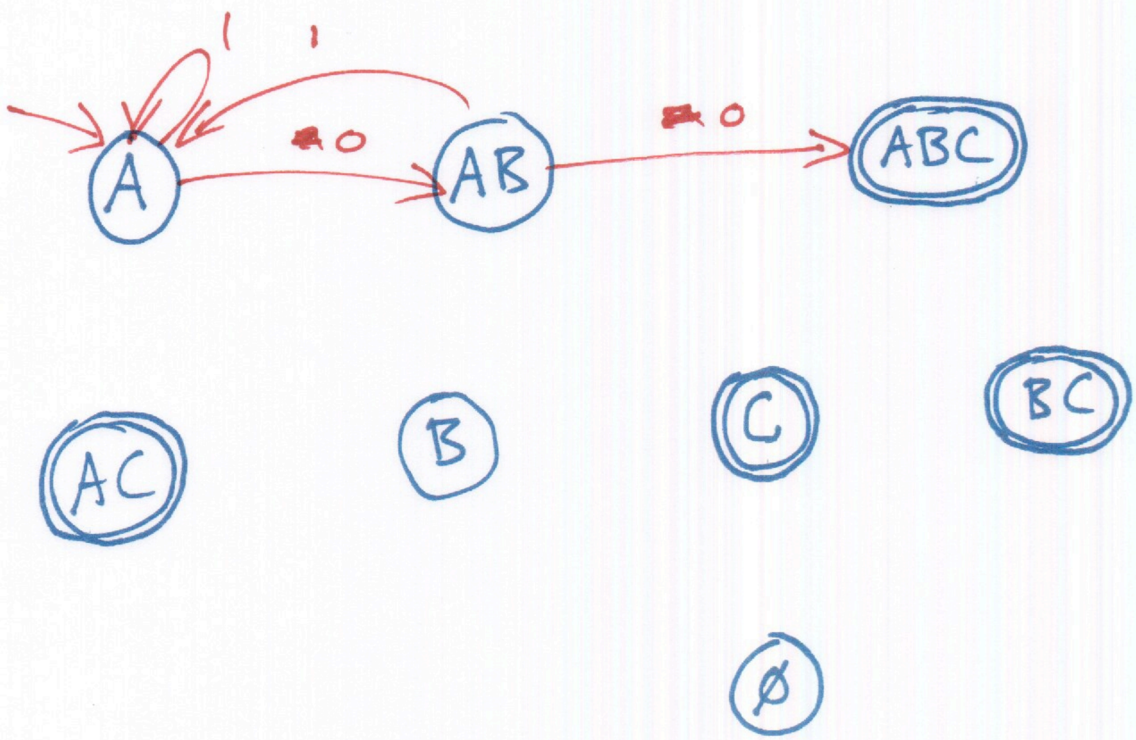
Let N = Number of states in NFA. What is the (worst case) number of states in the equivalent DFA?

Use Next Slide




28




28,1

## THEOREM

EVERY NONDETERMINISTIC FSM HAS AN EQUIVALENT DETERMINISTIC FSM.

"EQUIVALENT" = Recognizes the same language.

## PROOF BY CONSTRUCTION

Given a NFA, let's show how to build an equivalent DFA.

Let $M = (Q, \Sigma, \delta, q_0, F)$

This is the NFA we're given.

Construct

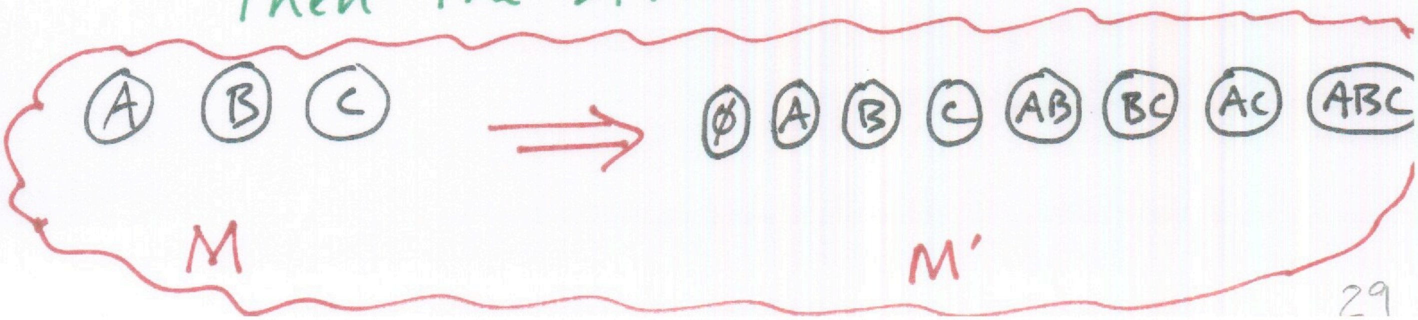
$$M' = (Q', \Sigma, \delta', q_0', F')$$

This is the DFA we're buiding.

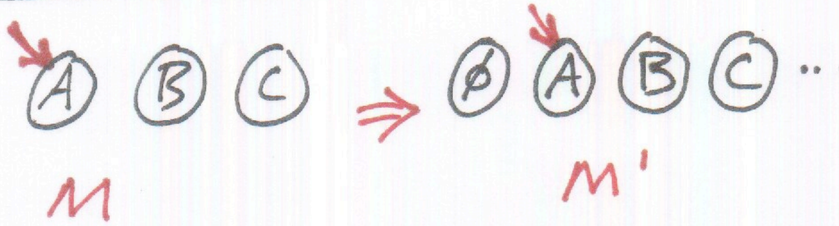
Where...

$$Q' = P(Q)$$

Assume NFA has $k$ states.
Then the DFA will have $2^k$ states.

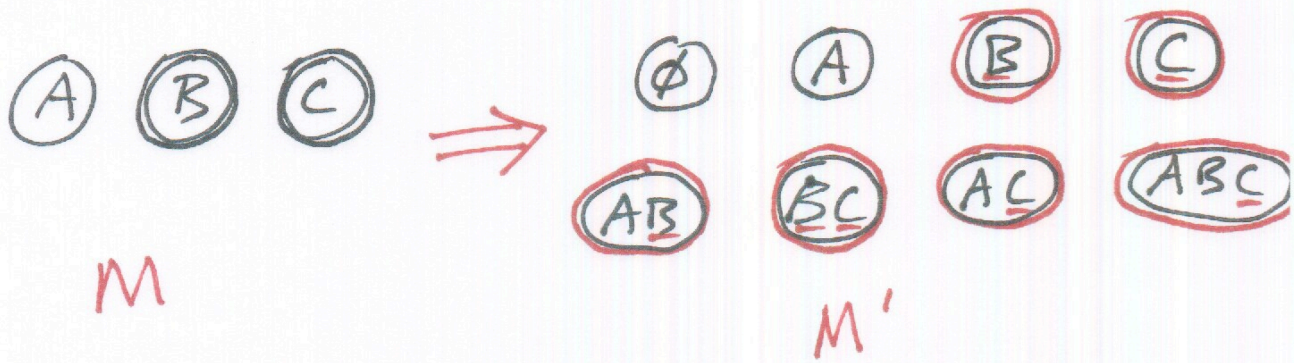


$q_0' = \{q_0\}$




$F' = \{R \in Q' \mid R \text{ contains an accept state from the NFA}\}$

"If the set contains a final state then it is final, too."

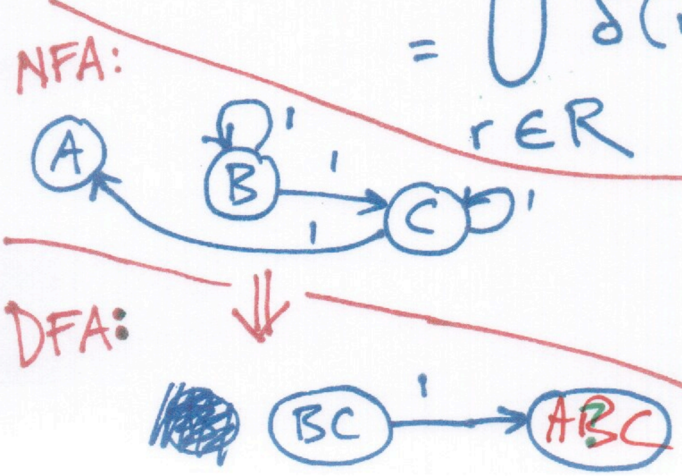



$\delta'(R, a) = \{q \mid q \in Q \text{ and } q \in \delta(r, a) \text{ for some } r \in R\}$

or equivalently:

$$= \bigcup_{r \in R} \delta(r, a)$$

a = an input symbol
R = a state in the DFA, i.e, a set of states in the NFA




$\delta(B, 1) = \{B, C\}$

$\delta(C, 1) = \{A, C\}$

$\delta(BC, 1) = \{B, C, A\}$

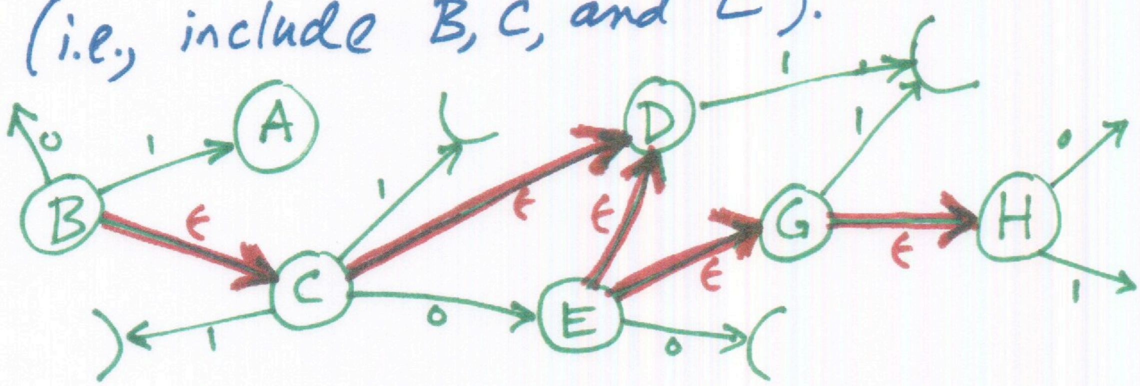
30

# BUT WHAT ABOUT $\epsilon$-EDGES?

Consider a state in the DFA that we're building.

BCE ← A state "R" in the DFA is a set of states from the NFA.

Look back at M, the NFA.
What states can we reach by going through $\epsilon$-edges?
Also include the states we're in.
(i.e., include B, C, and E).




## DEFINE "EPSILON-CLOSURE"

$$\mathbf{E}(R) = \{ q \in Q \mid q \text{ can be reached from a state in } R \text{ by following zero or more } \epsilon\text{-edges.} \}$$

## EXAMPLE

$$\mathbf{E}(\text{BCE}) = \{B, C, D, E, G, H\} = \text{BCDEGH}$$

31

Modify the transition function:

$$\delta'(R,a) = \{q \in Q \mid q \in \mathbf{E}(\delta(r,a))\}$$

for some $r \in R$

Also, modify the start state in the constructed DFA:

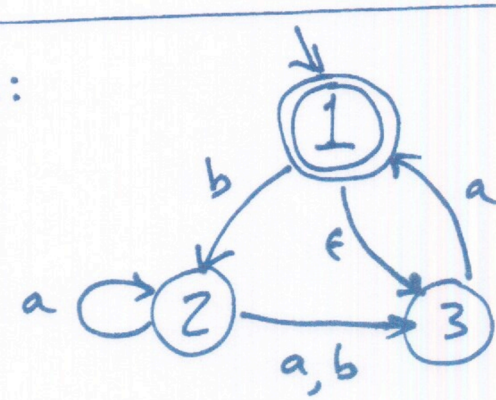
$$q_0' = \mathbf{E}(\{q_0\})$$

$\boxed{\text{END OF PROOF}}$

32

EXAMPLE: CONVERT AN NFA TO A DFA.

NFA:




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

What states in the DFA?

$Q' = \{ \quad \downarrow \quad \} = P(Q)$

RED

DFA:

$\emptyset$ 1 2 3 12 13 23 123

Start State?

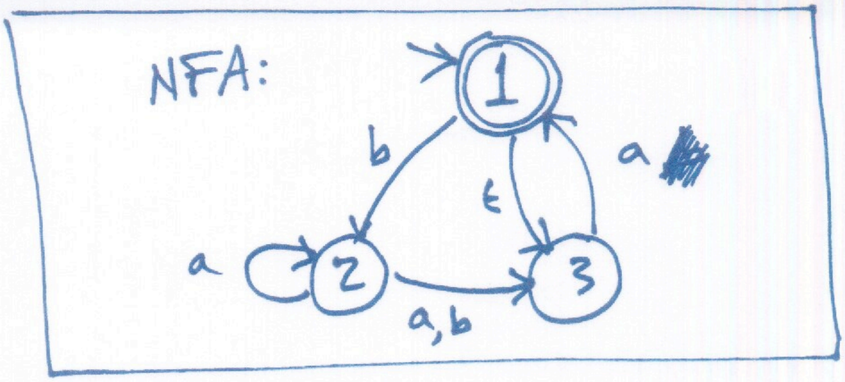
$q_0' = E(\{1\}) = \{1, 3\}$

$E(1) = 13$

ACCEPT STATES? ANY THAT CONTAIN 1

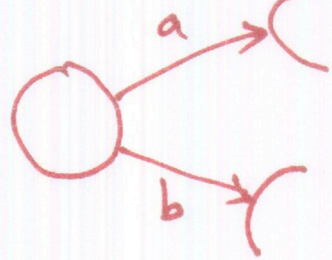
$F' =$ 1 12 13 123

33




TRANSITION FUNCTION?

FOR EACH STATE, WE NEED TWO EDGES:




$\delta'(1, a) = \emptyset$ Since no edges labeled "a" leave 1

$\delta'(1, b) = 2$ From 1 we can get to 2, but no further with ε-edges

$\delta'(2, a) = 23$

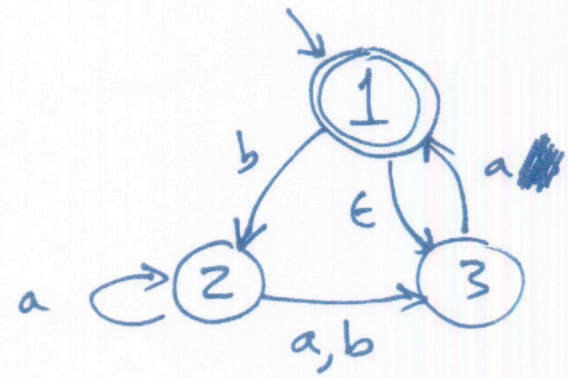
$\delta'(2, b) = 3$

$\delta'(3, a) = 13$ Can get to 1 but can get to 3 by following ε-edge.

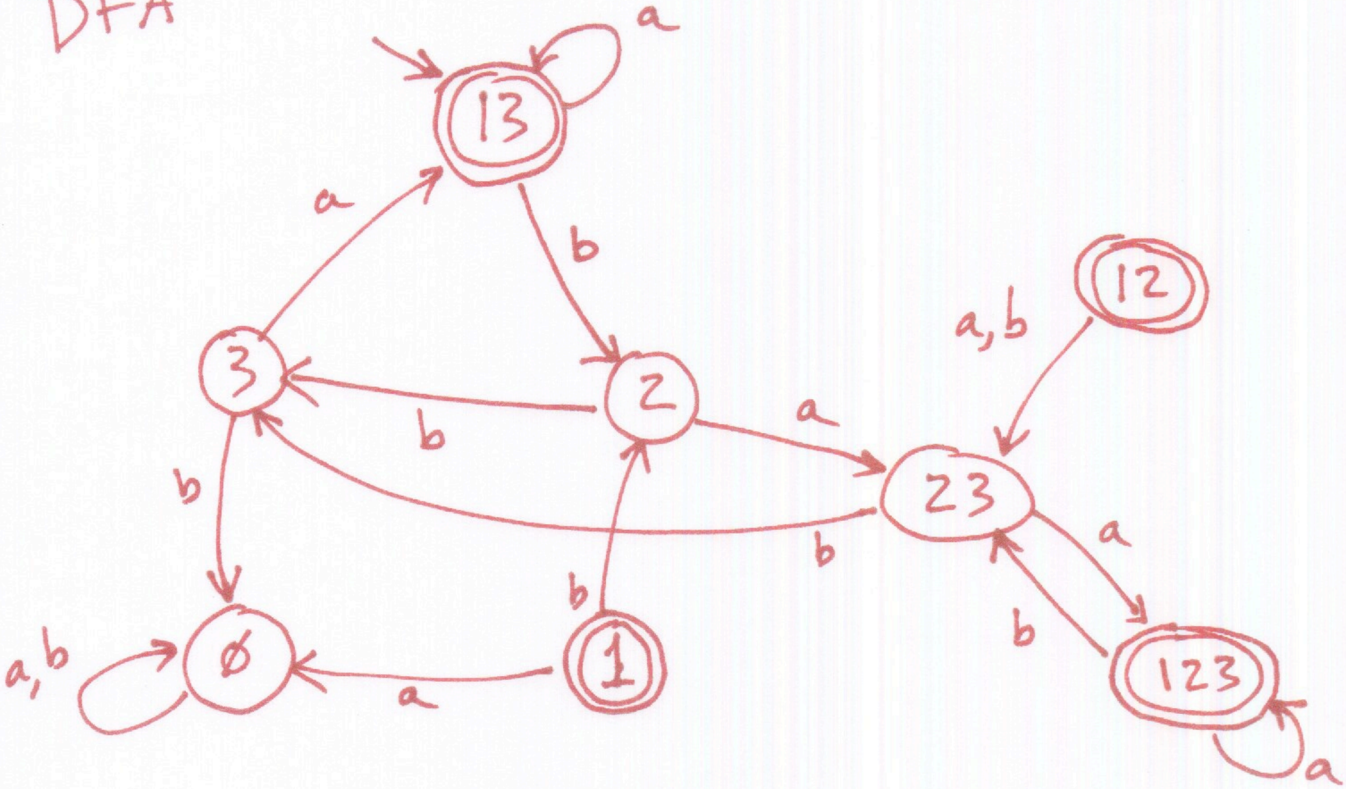
$\delta'(3, b) = \emptyset$

34

NFA:




DFA




NOTE: 1 AND 12 ARE UNREACHABLE.

THEY CAN BE REMOVED.

35

THEOREM

THE CLASS OF REGULAR LANGUAGES IS "CLOSED" UNDER UNION.

"CLOSURE" OF A LANGUAGE.

PROOF

ASSUME $A_1$ AND $A_2$ ARE REGULAR LANGUAGES.

SHOW $A_1 \cup A_2$ IS REGULAR.

---

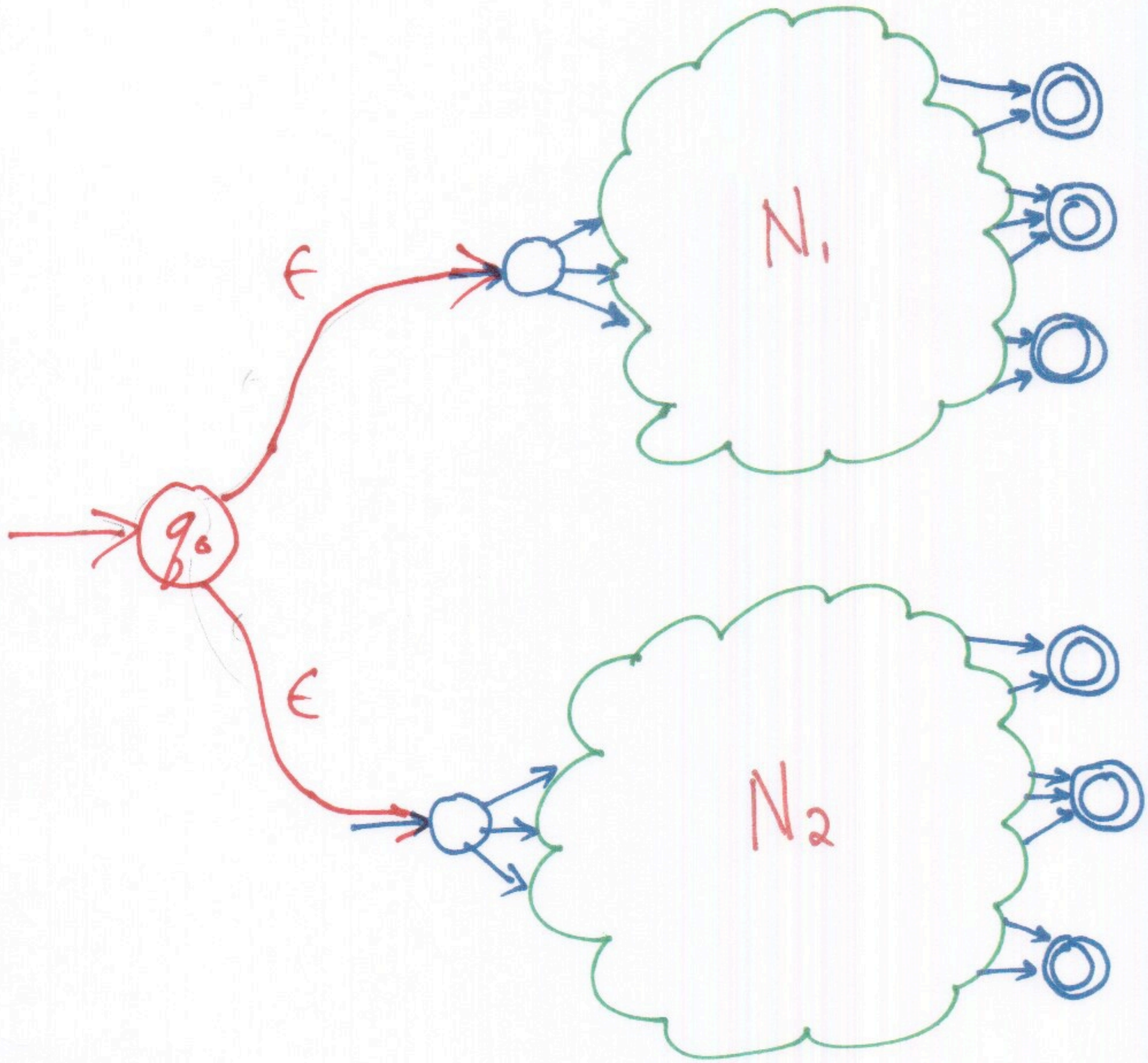
ASSUME NFA $N_1$ RECOGNIZES $A_1$

NFA $N_2$ RECOGNIZES $A_2$

COMBINE THEM;

TO BUILD: NFA $N$ TO RECOGNIZE $A_1 \cup A_2$

36




37

## FORMALLY:

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$

$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$

CONSTRUCT

$N = (Q, \Sigma, \delta, q_0, F)$

$Q = Q_1 \cup Q_2 \cup \{q_0\}$

$q_0$ is new start state

$F = F_1 \cup F_2$

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \underline{\text{if}}\ q \in Q_1 \\ \delta_2(q,a) & \underline{\text{if}}\ q \in Q_2 \\ \{q_1, q_2\} & \underline{\text{if}}\ q = q_0\ \underline{\text{and}}\ a = \epsilon \\ \{\} & \underline{\text{if}}\ q = q_0\ \underline{\text{and}}\ a \neq \epsilon \end{cases}$$

38

## THEOREM

THE CLASS OF REGULAR LANGUAGES IS CLOSED UNDER CONCATENATION.

Recall:

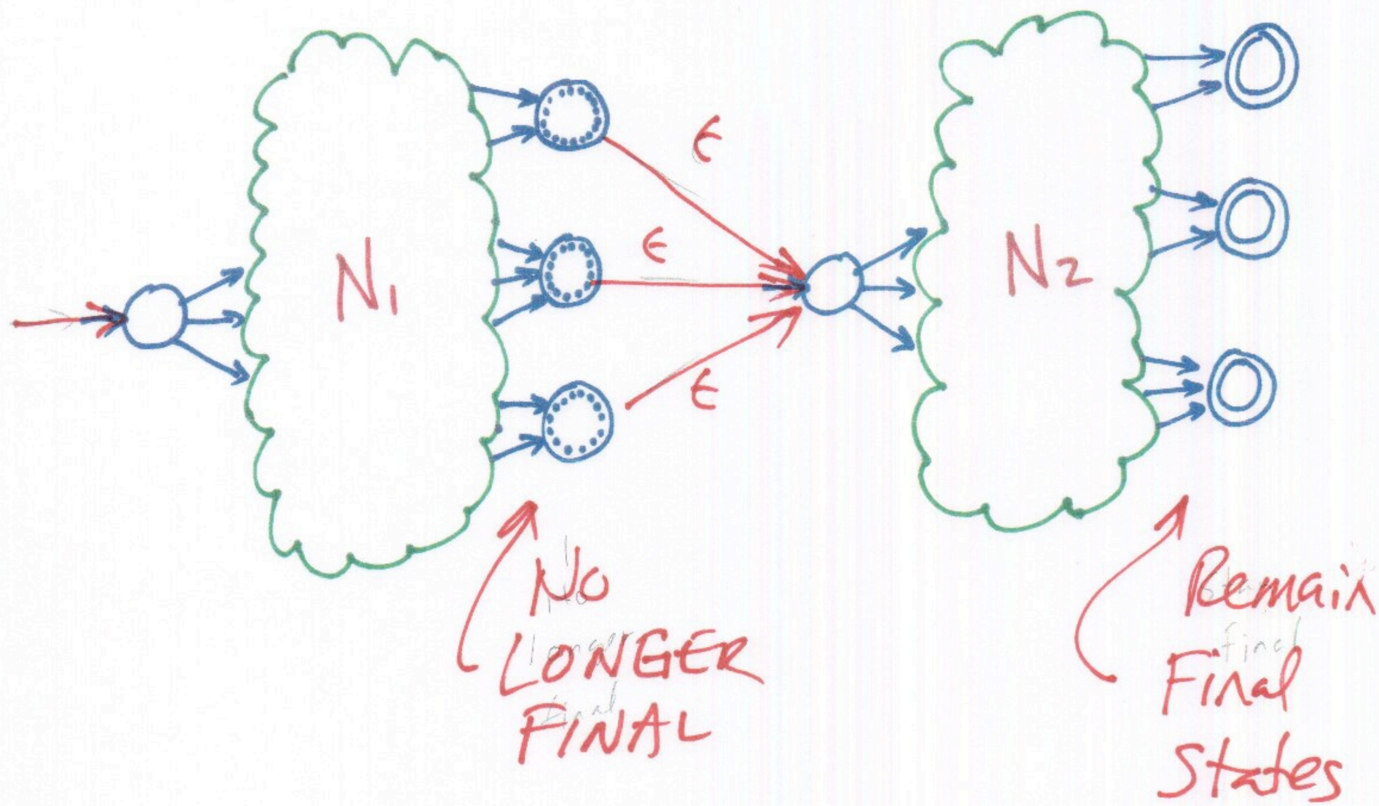
$w \in A_1 \circ A_2$ if $w = xy$ and $x \in A_1$
and $y \in A_2$.

## PROOF

SAME APPROACH.

Assume NFA $N_1$ recognizes $A_1$
and NFA $N_2$ recognizes $A_2$

Construct NFA $N$ to recognize $A_1 \circ A_2$




40

## FORMALLY

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$

$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$

Construct

$N = (Q, \Sigma, \delta, q_0, F)$

$Q = Q_1 \cup Q_2$

$q_0 = q_1$ the start state of $N_1$

$F = F_2$ the final states of $N_2$

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{if } q \in Q_1 \\ \delta_2(q,a) & \text{if } q \in Q_2 \\ \delta_1(q,a) \cup \{q_2\} & \underline{\text{if}}\ q \in F_1\ \underline{\text{and}}\ a = \epsilon \\ \delta_1(q,a) & \underline{\text{if}}\ q \in F_1\ \underline{\text{and}}\ a \neq \epsilon \end{cases}$$

41

THEOREM

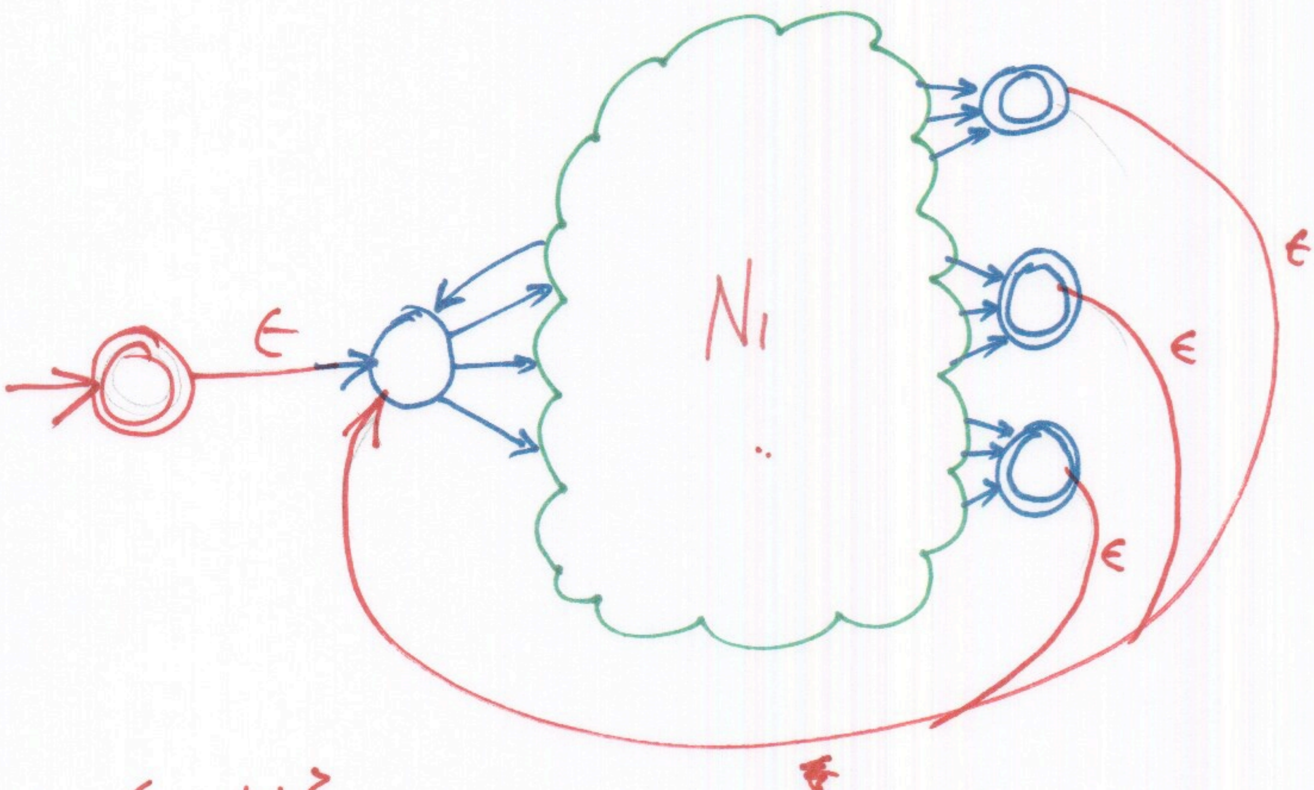
The class of regular languages is closed under "star".

CLOSURE

PROOF

Same idea.

$A^*$




$A = \{a, bb\}$

$A^* = \{\epsilon, a, aa, aaa, bb, bbbb,$
$abb, bba, bbaabbabb, \ldots\}$

42

DEFINITION

A "REGULAR EXPRESSION" IS...

(A recursive definition follows...)

$a$ is a Reg. Expression (where $a \in \Sigma$)

$R_1 \cup R_2$ is a Reg. Expression (where $R_1$ and $R_2$ are Regular Expressions)

Other Notation: $R_1 | R_2$

$R_1 \circ R_2$ is a Reg. Expression

Other Notation: $R_1 R_2$

$R_1{}^*$ is a Reg. Expression

$\epsilon$ is a Reg. Expression

$\emptyset$ is a Reg. Expression

$(R_1)$ is a Reg Expression

43

STAR BINDS TIGHTEST.

$$ab^* = a(b^*)$$
$$\neq (ab)^*.$$

CONCATENATION BINDS TIGHTER THAN UNION

$$a \circ b \cup c = (a \circ b) \cup c$$
$$\neq a(b \cup c).$$

OTHER NOTATIONS

| | | |
|---|---|---|
| $ab \mid c = (ab) \mid c$<br>$\neq a(b \mid c)$ | "UNION"<br>"OR" |
| $a^* = \{a\} = \{a\}^*$ | "STAR"<br>"CLOSURE" |
| $a^+ = aa^* = \{a\}^+$ | "ONE OR MORE" |
| $[a] = a \mid \epsilon = (a \cup \epsilon) = a?$ | "OPTIONAL" |

44

# Parsing Practice

$aa\ b \cup c\ aa\ b \cup c\ aa = ?$

$(aab) \cup (caab) \cup (caa)$

---

$aab \mid caab \mid caa$

$= (aab) \mid (caab) \mid (caa)$

---

$d \cup ab^* cd^* = ?$

$d \mid ab^* cd^* = ?$

$= (d) \cup (a(b^*)c(d^*))$

$= d \mid (a(b^*)c(d^*))$

45

# EXAMPLE REGULAR EXPRESSIONS

Assume $\Sigma = \{a, b, c, d\}$

$a$

$\{a\}$

$abccb$

$\{abccb\}$

$ab \cup cd = ab | cd$

$\{ab, cd\}$

$a(b \cup c)d = a(b|c)d$

$\{abd, acd\}$

$ab^*c$

$\{ac, abc, abbc, abbbc, \ldots\}$

$a(b \cup \epsilon)c = a(b|\epsilon)c = a[b]c$

$\{abc, ac\}$

$\emptyset$

$\{\}$

$a(b \cup c)\emptyset$

$\{\}$

$\emptyset^*$

$\{\epsilon\}$

46

## REGULAR EXPRESSION

Each Regular Expression describes a Language. Which Language?

$L(R) = ?$

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

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

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

$L(R_1^*) = L(R_1)^*$

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

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

$L((R_1)) = L(R_1)$

Regular Languages are closed under Union, Concatenation, and Star.

$\Rightarrow$ Regular Expressions describe Regular Languages.

47

THEOREM

A LANGUAGE IS REGULAR IFF SOME REGULAR EXPRESSION DESCRIBES IT.

PREVIOUS DEFINITION

A LANGUAGE IS REGULAR IFF IT IS ~~DESCRIBED~~ RECOGNIZED BY SOME FINITE STATE MACHINE. (NFA=DFA).

GIVEN A REGULAR EXPRESSION, E, THE LANGUAGE IT DESCRIBES

$L(E)$ = See previous slide

So we are saying that the class of languages recognized by DFA's, NFAs, and Reg. Expressions is the same! All have equivalent "power"!

48

## LEMMA 1:

IF A LANGUAGE IS DESCRIBED BY A REGULAR EXPRESSION, THEN IT IS REGULAR.

PROOF #1: USE THE CLOSURE OF $\cup$ $*$ and $\circ$.

PROOF #2: FROM A REGULAR EXPRESSION, BUILD AN NFA TO RECOGNIZE IT.

## LEMMA 2:

IF A LANGUAGE IS REGULAR, THEN IT CAN BE DESRIBED BY A REGULAR EXPRESSION.

PROOF APPROACH:

- START WITH A DFA THAT RECOGNIZES IT.
- BUILD A GNFA (GENERALIZED NONDETERMINISTIC FINITE STATE AUTOMATON).
- REDUCE IT (details to follow).
- THIS YIELDS A REGULAR EXPRESSION.

48.1

## LEMMA 1

- WE HAVE A REGULAR EXPRESSION. CALL IT R.
- CONVERT R TO A NFA.
- CONCLUDE THAT THE LANGUAGE MUST BE REGULAR.

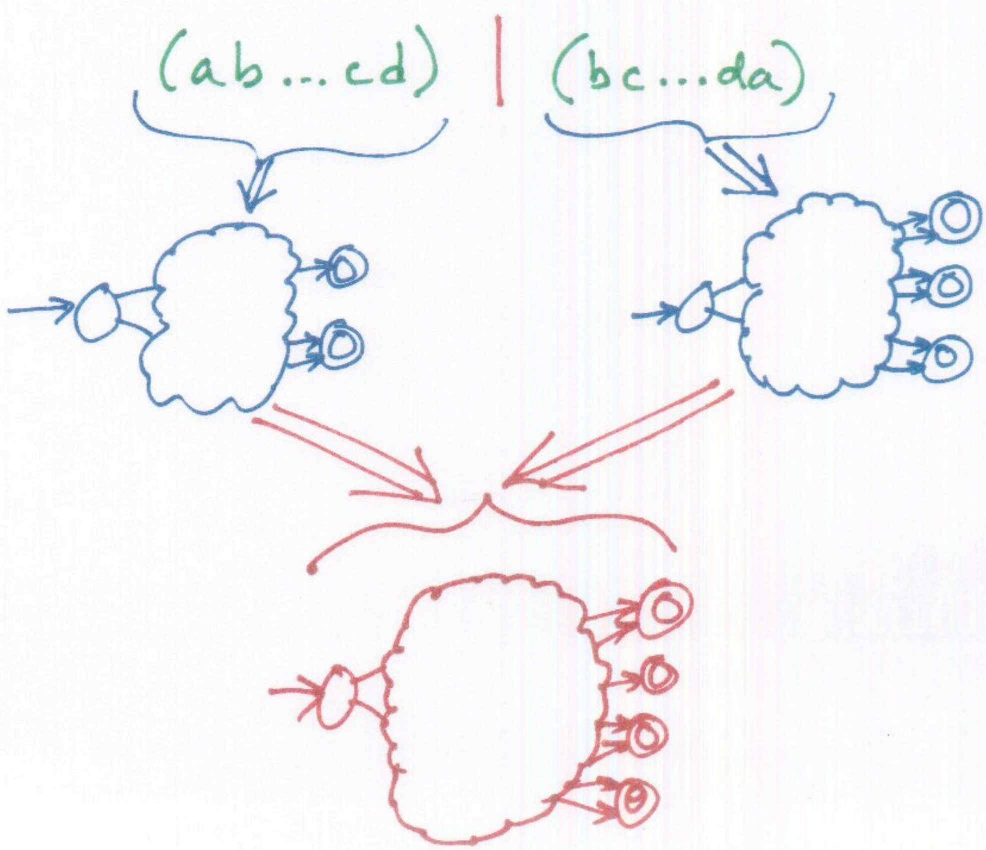
---

EVERY LARGE REGULAR EXPRESSION IS MADE OF SMALLER REGULAR EXPRESSIONS.

ASSUME WE CAN BUILD THE NFAs FOR SMALLER REGULAR EXPRESSIONS.

SHOW HOW TO BUILD THE NFA FOR LARGER REGULAR EXPRESSIONS.

INDUCTIVE PROOF! (STRUCTURAL INDUCTION)




49

# DEFINITION OF REGULAR EXPRESSIONS

$R = a$

$R = R_1 | R_2$

$R = R_1 \circ R_2$

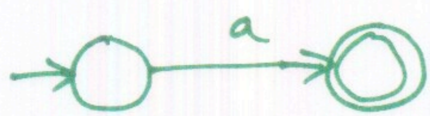
$R = R_1{}^*$

$R = \epsilon$

$R = \emptyset$

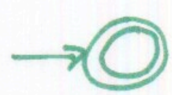
---

$R = a$




---

$R = \epsilon$



---

$R = \emptyset$

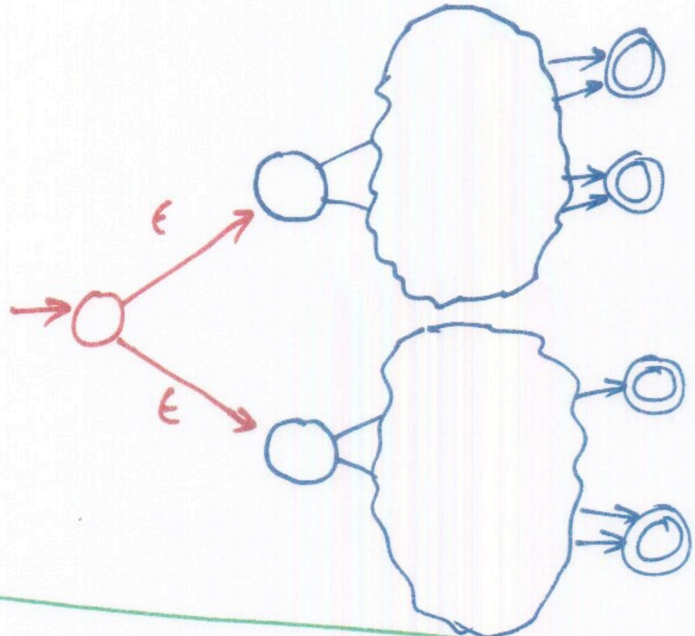
---

$$\left.\begin{array}{l} R = R_1 | R_2 \\ R = R_1 \circ R_2 \\ R = R_1{}^* \end{array}\right\}$$

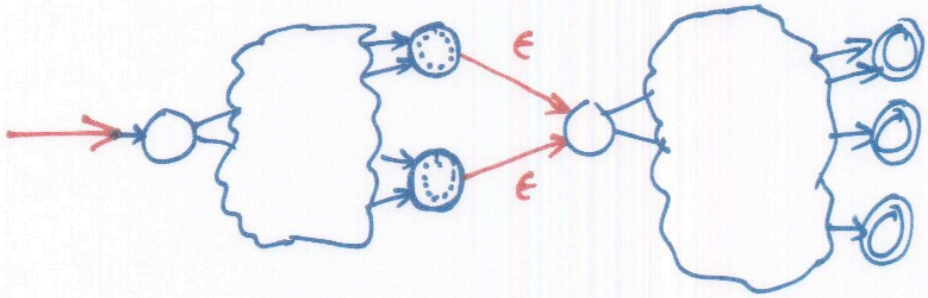
Same construction used in proof of closure.

50

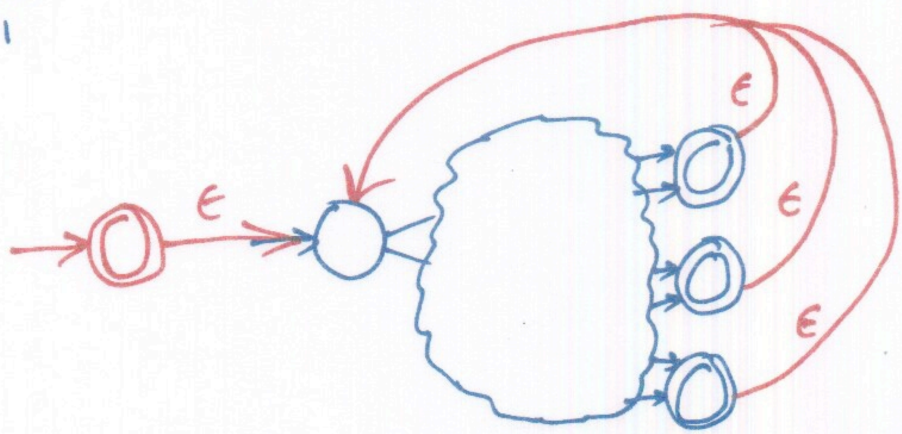
$R_1 | R_2$




$R_1 \circ R_2$




$R_1^*$




51

LEMMA 2

IF A LANGUAGE IS REGULAR, THEN IT CAN BE DESCRIBED BY A REGULAR EXPRESSION

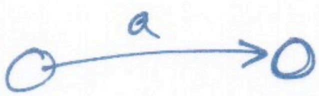
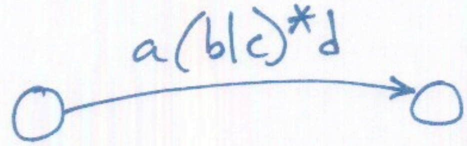
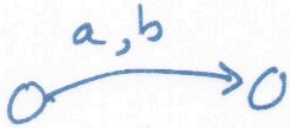
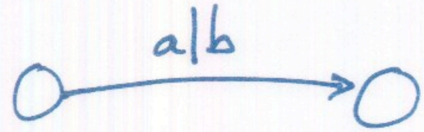
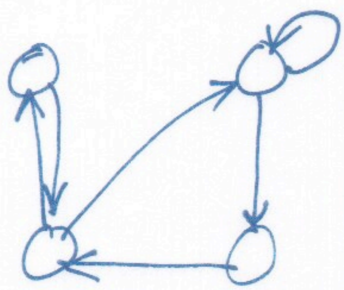
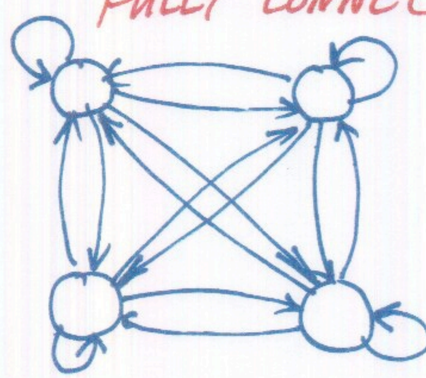
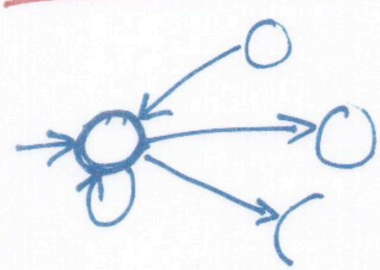
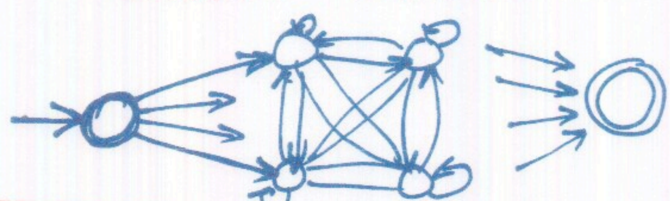
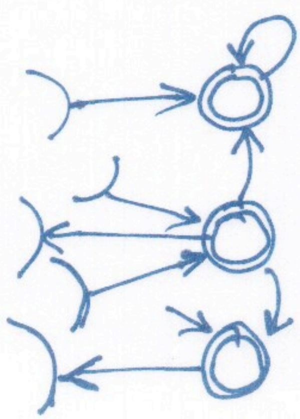
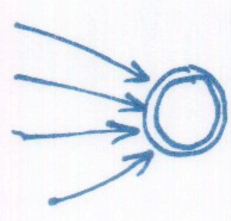
GENERALIZED NONDETERMINISTIC FINITE AUTOMATON (GNFA)

LIKE A N.F.A.

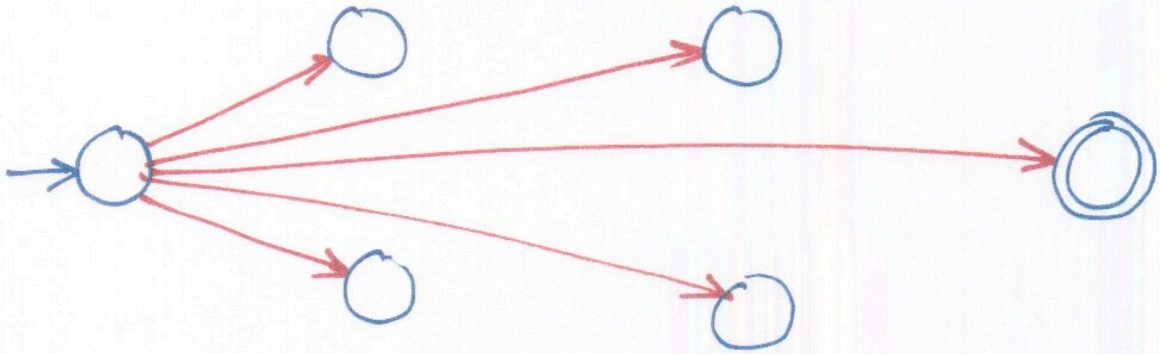
<u>EXCEPT:</u>

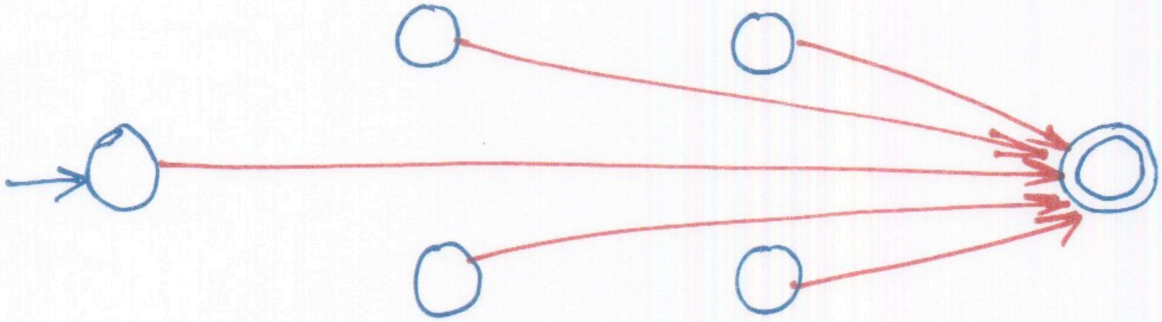
- EDGES ARE LABELED WITH <u>REGULAR</u> <u>EXPRESSIONS</u>!
- ONLY <u>ONE</u> ACCEPT STATE
- THERE IS EXACTLY ONE EDGE FROM EVERY STATE TO EVERY OTHER STATE. (INCLUDING AN EDGE TO SAME STATE
- EXCEPT: NO EDGES GOING TO THE START STATE.
- EXCEPT: NO EDGES GOING OUT OF THE ACCEPT STATE.

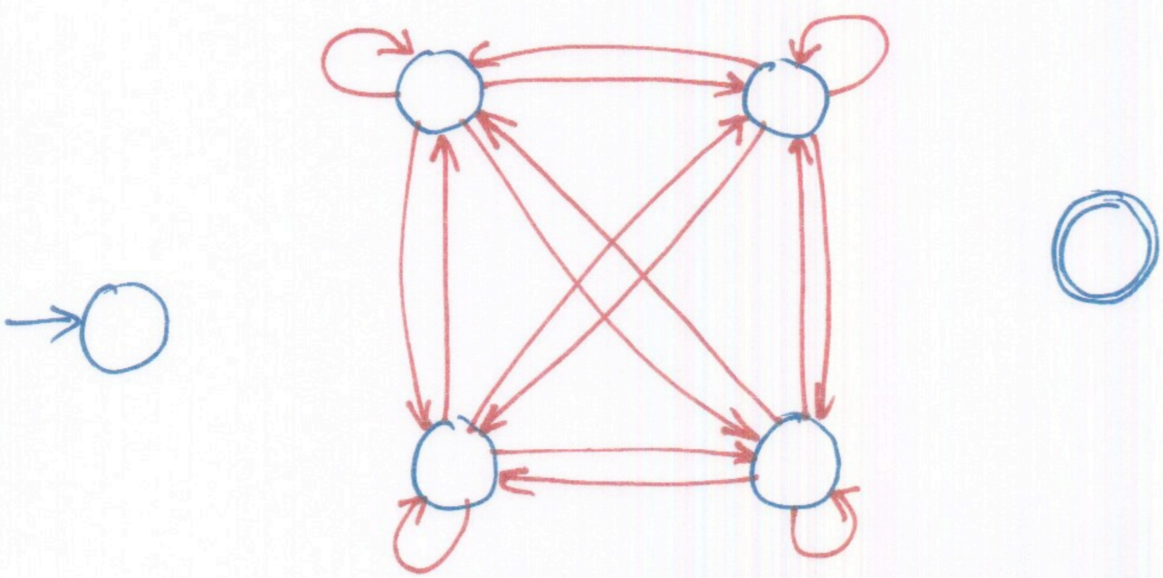
52

| NFA | GNFA |
|---|---|
|   |   |
|   | ONLY ONE EDGE BETWEEN STATES   |
|  | FULLY CONNECTED  |
|  | NO EDGES TO START STATE  |
|  | ONLY ONE FINAL STATE; NO EDGES OUT OF IT.  |

53



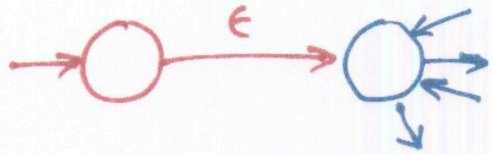




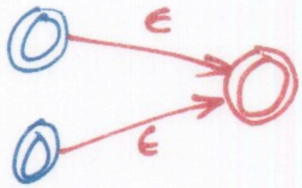
54

# STEP 1 CONVERT FROM DFA TO GNFA.

Add a start state.

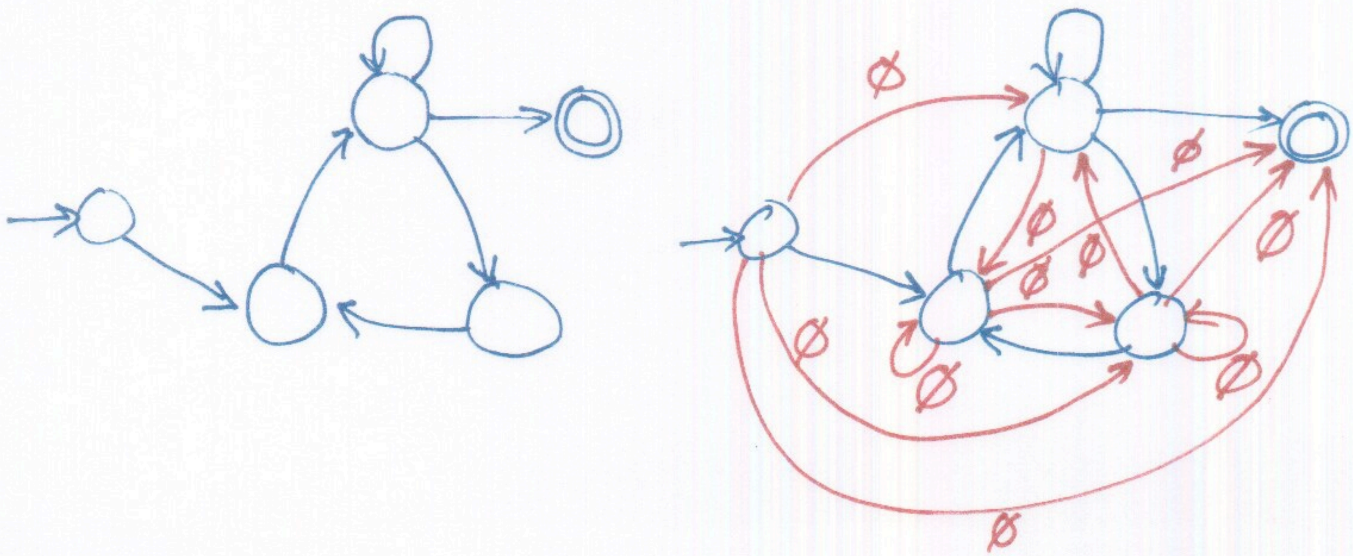



Add a new ACCEPT STATE.



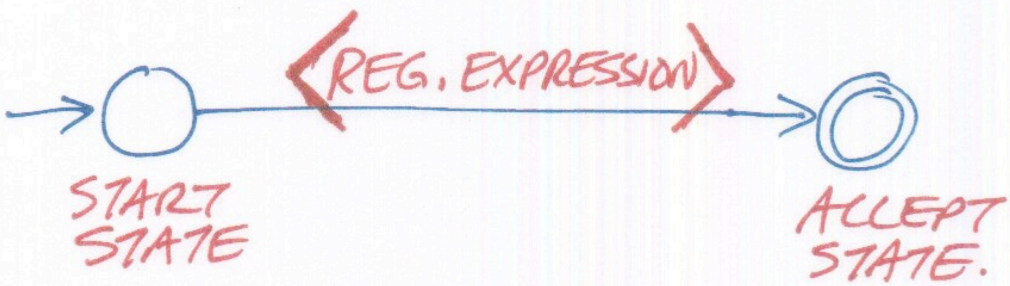

Eliminate multiple edges with UNION.


Add missing edges (with ∅).



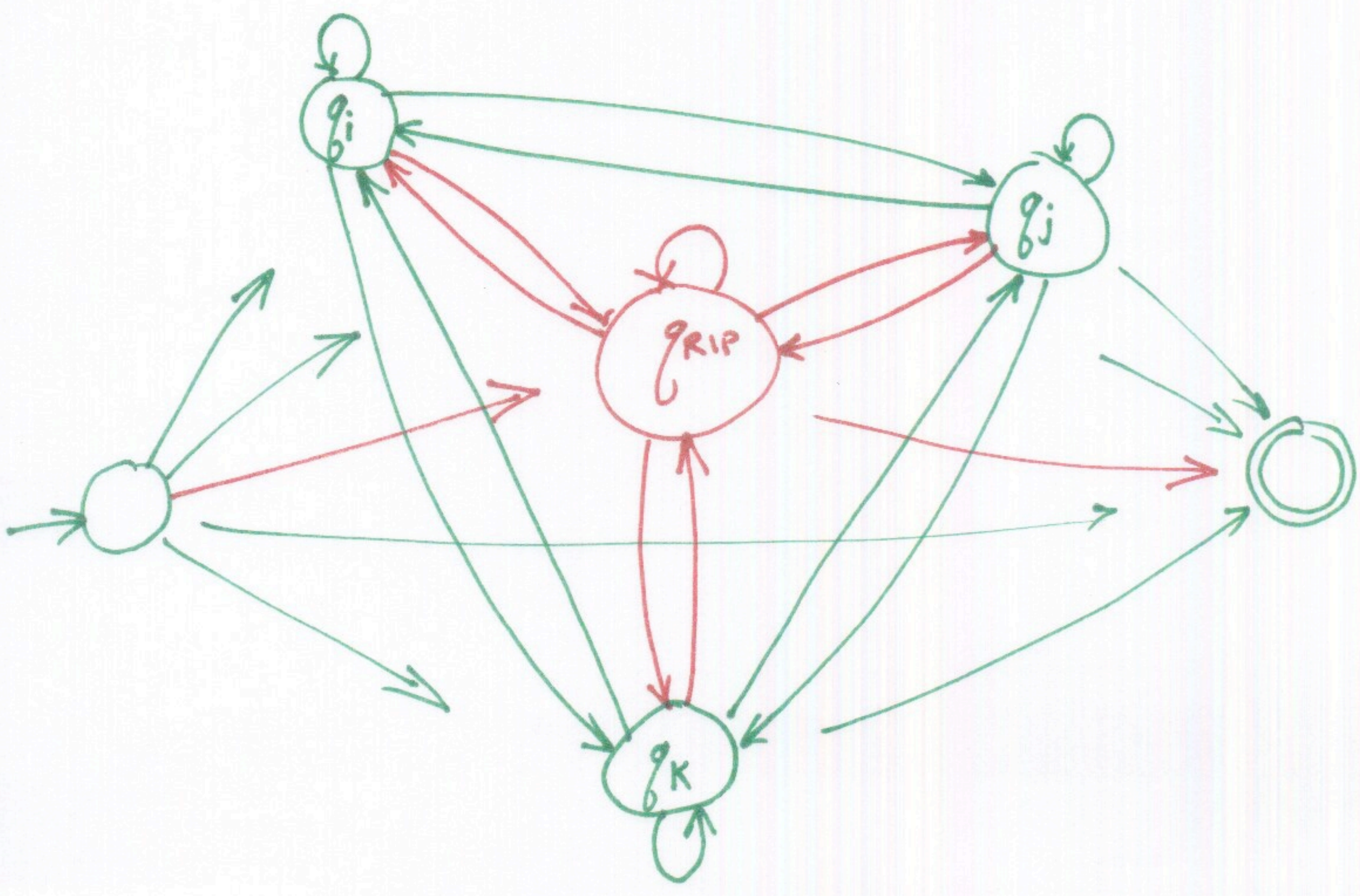
STEP 2:

- CHOOSE A STATE.
- REMOVE IT.
- MODIFY THE MACHINE SO IT STILL ACCEPTS THE SAME LANGUAGE.
- REPEAT UNTIL THERE ARE ONLY 2 STATES LEFT.




- WE HAVE NOW CONVERTED OUR DFA ~~NFA~~ INTO AN EQUIVALENT REGULAR EXPRESSION THAT RECOGNIZES THE SAME LANGUAGE.

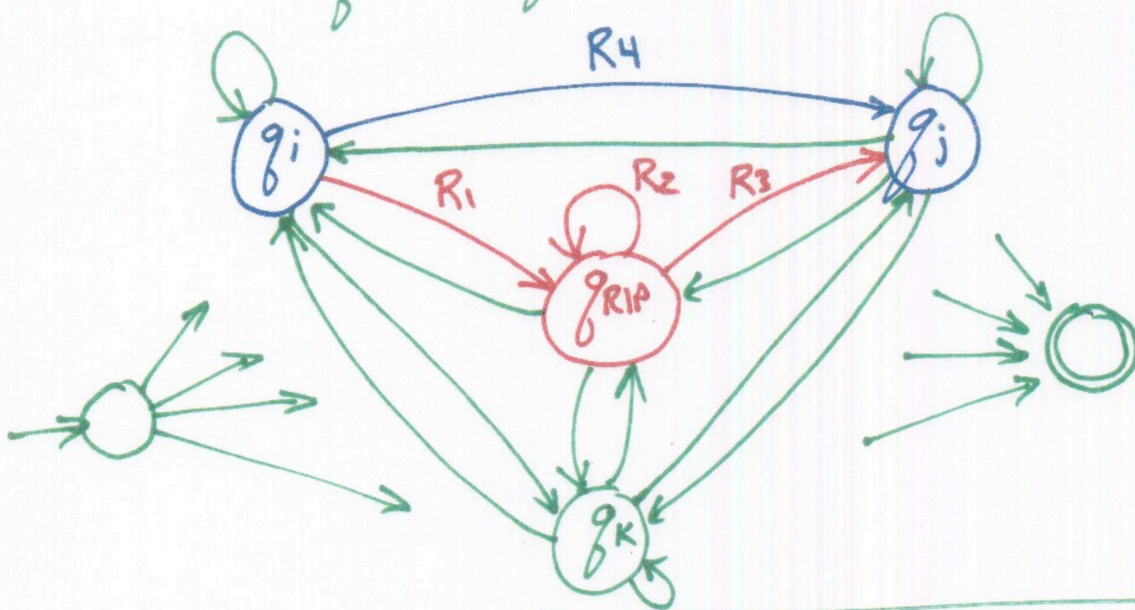
- SELECT A STATE AT RANDOM (BUT DON'T SELECT THE START OR ACCEPT STATES)
- CALL IT $q_{rip}$
- RIP THIS STATE OUT OF THE GNFA.
- REMOVE $q_{RIP}$ AND ALL EDGES TO/FROM IT.
- MODIFY THE OTHER EDGES SO THAT THE RESULTING MACHINE STILL ACCEPTS THE SAME LANGUAGE.

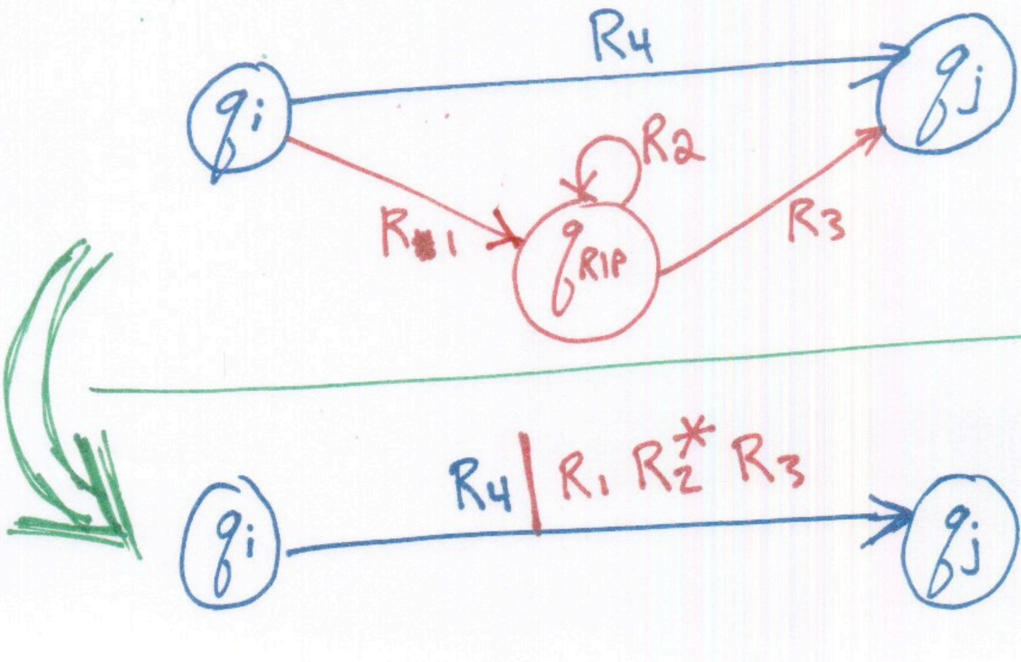



57

- WE HAVE TO MODIFY <u>EVERY</u> REMAINING EDGE.
- CONSIDER: 

- WE COULD ALSO HAVE GOTTEN FROM $q_i$ to $q_j$ BY GOING THROUGH $q_{RIP}$.




57.1

EXAMPLE

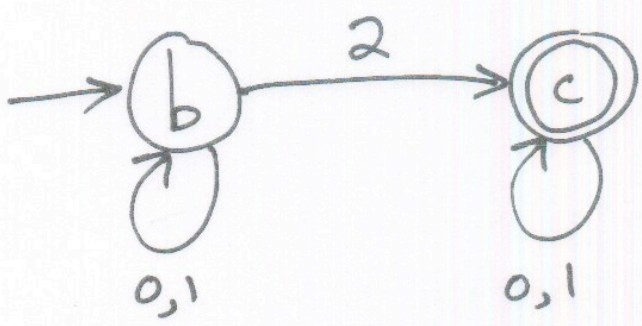
Will want to simplify our Regular Expressions.

$$\epsilon R = R\epsilon = R$$

$$\emptyset R = R\emptyset = \emptyset$$

$$\emptyset | R = \{\} \cup R = R$$

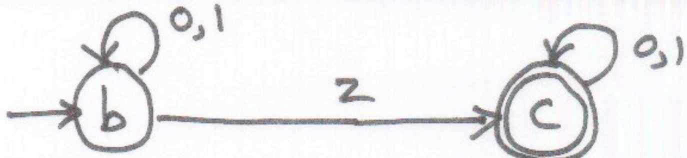
$$\Sigma = \{0, 1, 2\}$$



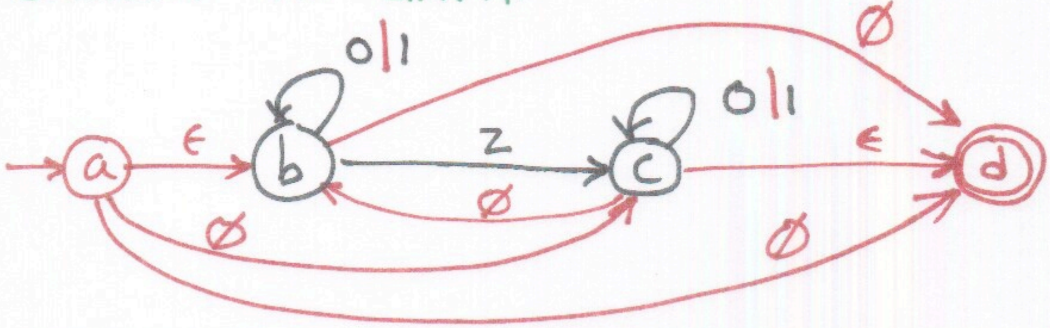

DESIRED ANSWER:

$$(0|1)^* 2 \ (0|1)^*$$

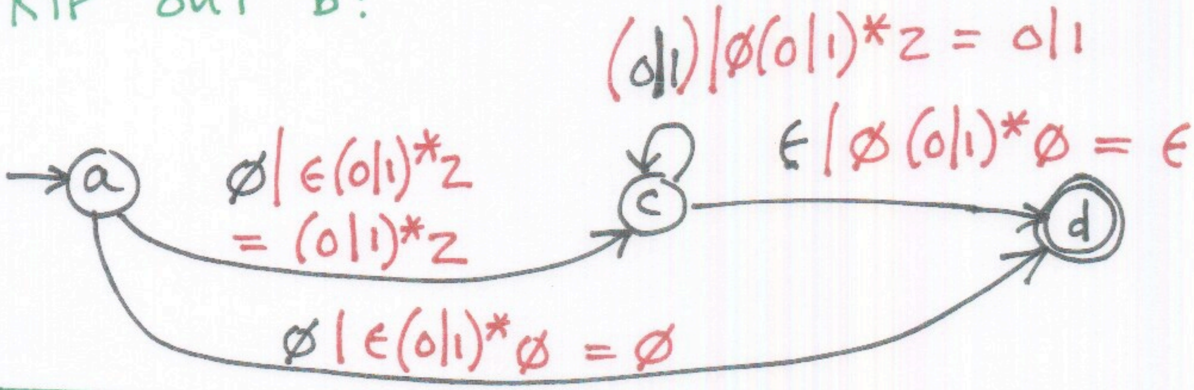
58



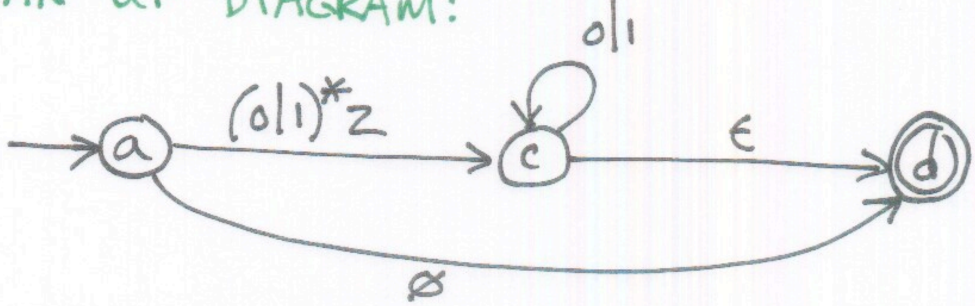

CONVERT TO GNFA:



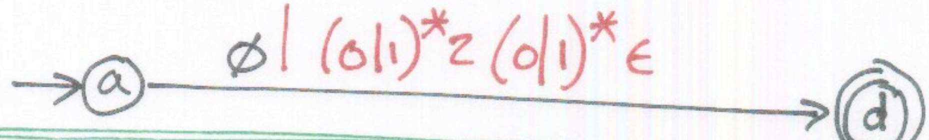

RIP OUT b:



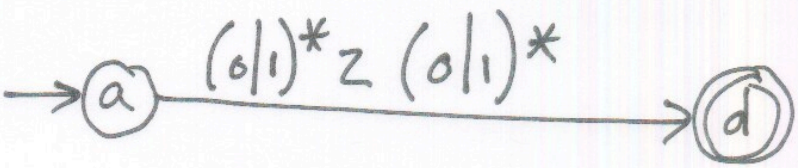

CLEAN UP DIAGRAM:




RIP OUT c:




SIMPLIFY:




59

# NON-REGULAR LANGUAGES

$B = \{ 0^N 1^N \mid N \geq 0 \}$

$C = \{ w \mid w$ has an equal number of 0's and 1's $\}$

$D = \{ w \mid w$ has an equal number of "01"s and "10"s $\}$

SURPRISE! This <u>is</u> regular.

---

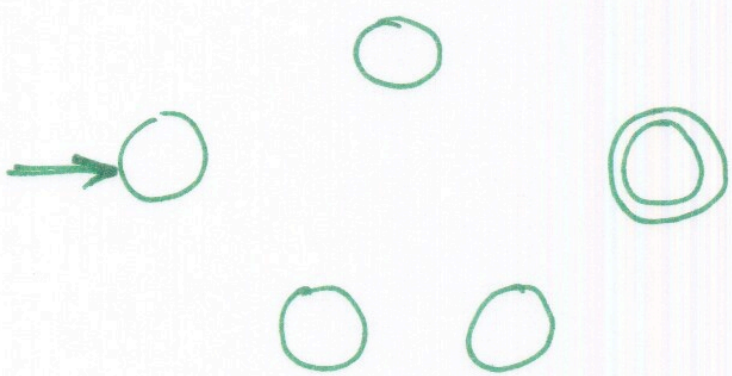
HOW CAN WE <u>PROVE</u> A LANGUAGE IS NOT REGULAR?

$$\boxed{\text{THE PUMPING LEMMA!}}$$

60

IMAGINE A F.S.M. THAT GENERATES REALLY LONG STRINGS.

- The F.S.M. will have a small number of states.



$|Q| = n$
$= 5$

How can we generate a long string?

TAKE A CYCLIC PATH

How long can the string be without containing a cycle?

$|s| < n$

Any string of length **5** (or longer) must take a cycle.

KEY IDEA:

"If you can go around the cycle once, then you can go around it two times, and that string MUST ALSO BE IN the language."

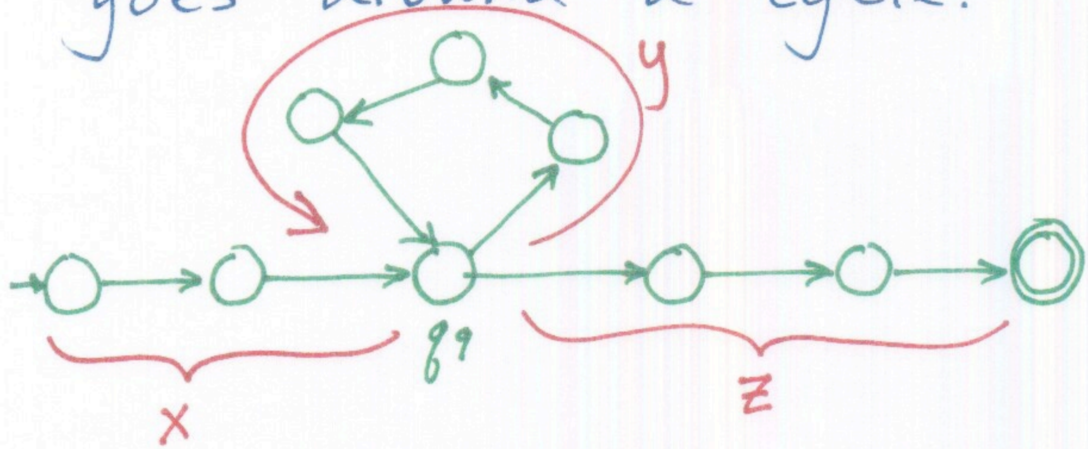
(You can also go around the cycle lots of times. ("i" times))

"You can also skip the cycle, and that string MUST ALSO BE in the language."

(You can go around the cycle i = 0 times.)

62

Look at a string "s" that goes around a cycle.




String $s = xyz$

**But Note:** All strings of the form

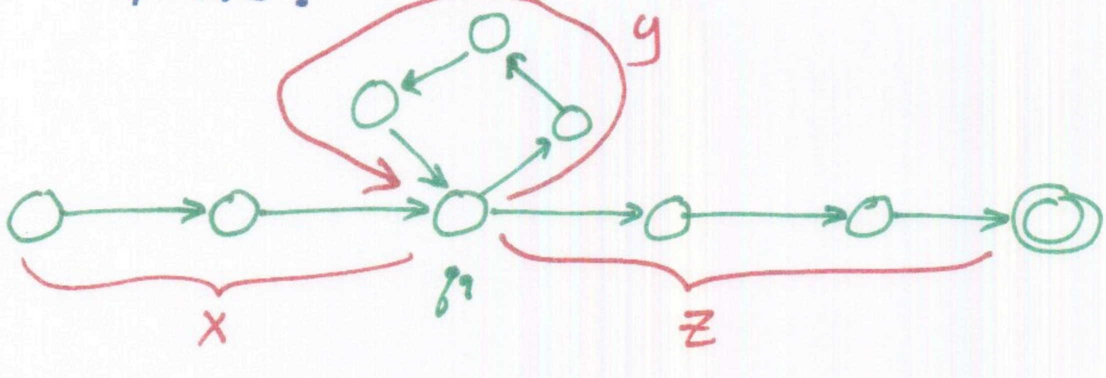
$$s = xy^iz$$

are in the language.

$$s = xy^iz \in A \text{ for } i \geq 0$$

63

What can we say must be true?




If A is a regular language, and if s is a sufficiently long enough string

i.e., $|s| \geq p$

The PUMPING LENGTH. Depends on the language, not any specific F.S.M. Long enough that some FSM will have a cycle.

Then s can be divided up

$s = xyz$

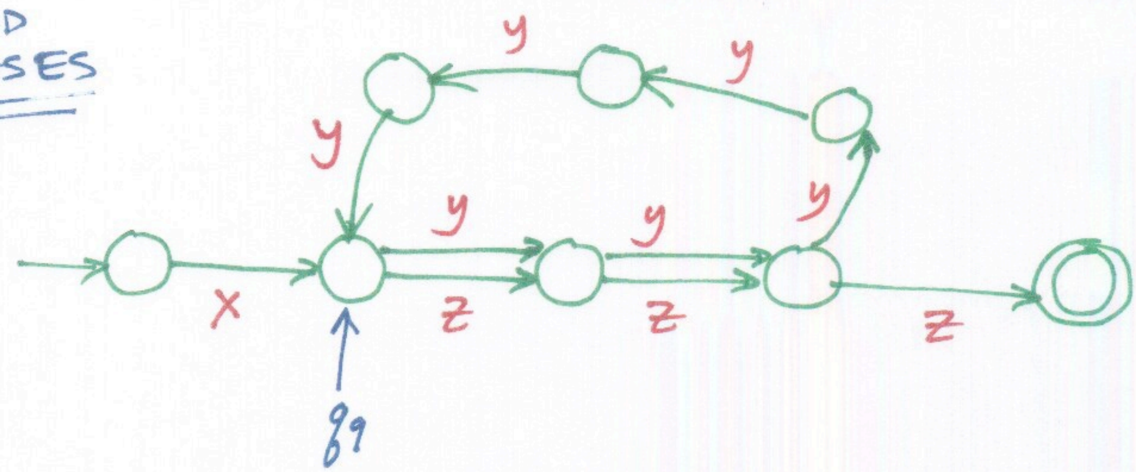
Such that:

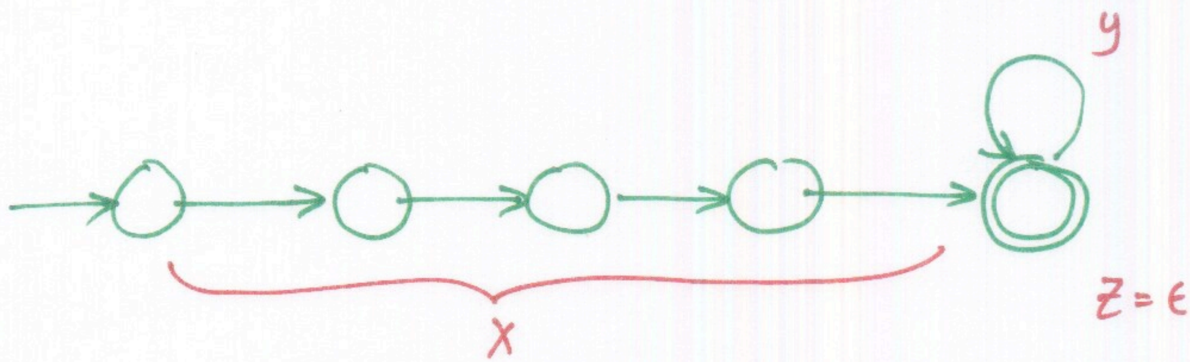
$xy^iz$ is in the language, for $i \geq 0$

$|y| > 0$ ← The cycle has at least one edge in it.

$|xy| \leq p$ ← You have to hit the cycle before the string gets longer than p.

ODD CASES




$|y| > 0$ just barely

$|xy| \le p$ just barely.

$p = 5$

65

NOTE:

We are saying:

$\boxed{\text{If the language is regular, then there must be a length [let's call it the PUMPING LENGTH, "p"] such that: All strings longer than p can be "pumped."}}$

- This pumping length is a property of the language, and not of any specific Finite State Machine.
- If the language is regular, there must be a F.S.M.; Choose any one.

If it makes you feel better, choose the MINIMAL Finite State Machine, i.e., the one with the fewest states.

66

# PUMPING LEMMA

If $A$ is a REGULAR LANGUAGE, then $A$ has a pumping length $p$ such that *any* string $s$ may be divided into 3 pieces

As long as: $|s| \geq p$

$$s = x\,y\,z$$

such that all these conditions hold.

**CONDITION 1**

$xy^iz \in A$ for every $i \geq 0$

**CONDITION 2**

$|y| > 0$

**CONDITION 3**

$|xy| \leq p$

67

USE THE "PUMPING LEMMA" TO PROVE THAT A LANGUAGE "A" IS NOT REGULAR.

BY CONTRADICTION:

- ASSUME A IS REGULAR.
- It HAS A PUMPING LENGTH. SO CALL IT "p".
- ALL STRINGS LONGER THAN p CAN BE PUMPED. $|s| \geq p$
- FIND A STRING "s" IN A. SUCH THAT $|s| \geq p$

Just need to find one string.

- DIVIDE s INTO xyz.
- SHOW THAT $xy^iz \notin A$ FOR SOME i.
- THEN CONSIDER <u>ALL</u> WAYS THAT s CAN BE DIVIDED INTO xyz.
- SHOW THAT NONE OF THESE CAN SATISFY ALL THE 3 PUMPING CONDITIONS AT THE SAME TIME.
- S CANNOT BE PUMPED ⇒ <u>CONTRADICTION</u>

18

**EXAMPLE**

Let $B = \{ 0^n 1^n \mid n \geq 0 \}$
Prove $B$ is not regular.

Assume that $B$ is regular.

$B$ must have a pumping length.
Let "$p$" be the pumping length.

The string we will use to get the contradiction is

$$s = 0^p 1^p$$

Lets divide $s$ into pieces $xyz$.

CASE 1: The "$y$" is in the zeros part.

CASE 2: The "$y$" is in the ONES part.

CASE 3: The "$y$" has zeros and ones

$$0^7 1^7 = 00\underbrace{0000}_{y}\,\underbrace{011}_{y}\,\underbrace{1111}_{y}\,1$$

CASE 1: $0\underbrace{0000}_{y}0\,11111$

So $xy^iz$ must be in B.

$xy^2z = 0\underbrace{0000}_{y}\underbrace{0000}_{y}0\,11111$

CASE 2: $00000\,1\underbrace{111}_{y}1$

$xy^2z = 00000\,1\underbrace{111}_{y}\underbrace{111}_{y}1$

CASE 3: $000\underbrace{0011}_{y}111$

$xy^2z = 000\underbrace{0011}_{y}\underbrace{0011}_{y}111$

Not of the form $0^n1^n$

ALSO RECALL CONDITION 3

$|xy| \leq p$

$s = 0^p1^p = \overbrace{\underbrace{00}_{x}\underbrace{000}_{y}}^{p}\,11111$

So cases 2 and 3 could be ruled out that way too!

70

## EXAMPLE

Let $F = \{ww \mid w \in \{0,1\}^*\}$

Show $F$ is not regular.

## PROOF

- Assume it is regular.
- Let $p$ be the pumping length of $F$.
- Let's use $s = 0^p 1 0^p 1$

- Now split $s$ into 3 pieces

$$s = x\,y\,z$$

condition 2: $|y| > 0$

condition 3: $|xy| \le p$

$p=7$: $0^7 1 0^7 1 = 0\underbrace{000000}_{y}0 \,\vdots\, 100000001$

- So $y$ must be all zeros.

- So $xy^2z$ is not in $F$.

- Contradiction!

F is <u>not</u> regular.

71

# REGULAR LANGUAGES AND FINITE STATE MACHINES

SOME QUESTIONS THAT WE CAN ANSWER...

"DECIDABLE QUESTIONS"
CAN WRITE A PROGRAM.
THE PROGRAM WILL ALWAYS TERMINATE.

GIVEN A F.S.M., WHAT IS THE MINIMAL EQUIVALENT F.S.M?

DO TWO FSMs ACCEPT THE SAME LANGUAGE?
(MINIMIZE EACH OF THEM & COMPARE THE GRAPHS.)

IS THE LANGUAGE EMPTY? OR INFINITE?

ARE TWO REGULAR EXPRESSIONS EQUIVALENT?

ALMOST EVERY PROBLEM ABOUT REGULAR LANGUAGES IS DECIDABLE!

... BUT MAY BE NP-COMPLETE, i.e., EXPONENTIAL IN THE NUMBER OF STATES.

72

# SUMMARY

Regular Expressions

$01(11011)^*00(11|00)11$

$\text{Letter}\ (\text{Letter} \cup \text{Digit})^*$

Regular Expressions

=

Regular Languages

=

Finite State Machines

DFA = NFA