

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

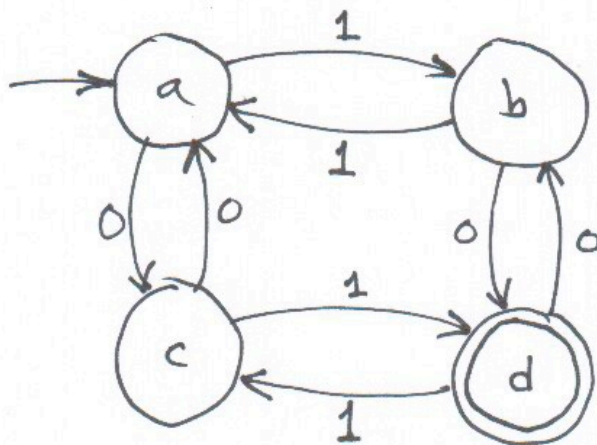
Limited Memory

Finite & usually quite small.

REGULAR LANGUAGES

REGULAR EXPRESSIONS

FINITE STATE MACHINE EXAMPLE



$\{0, 1\}^*$

1011001010

~~.100~~

STATES (NODES)

TRANSITIONS (EDGES)

STARTING STATE

Always exactly one starting state
"initial state"

ACCEPTING 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 ALL SYMBOLS IN STRING.
- DO YOU END UP IN AN "ACCEPTING" STATE OR NOT?
- THE SET OF STRINGS THAT ARE ACCEPTED?
- OTHERS ARE "REJECTED."

FORMAL DEFINITION OF FINITE STATE MACHINE

Described by a 5-tuple:

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

Q = Set of States

Finite Number of states

Σ = Alphabet, a Finite Set of Symbols

δ = The TRANSITION FUNCTION

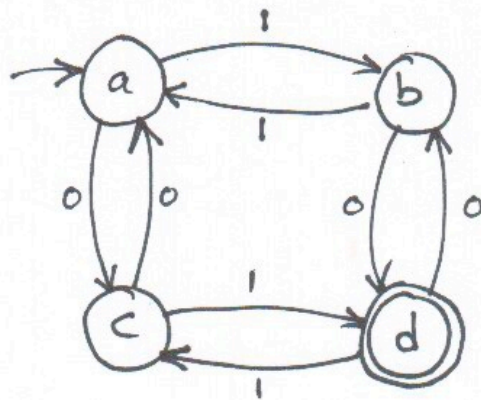
$$\delta: Q \times \Sigma \rightarrow Q$$

q_0 = The STARTING STATE

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

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

$$F \subseteq Q$$



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

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

$$q_0 = a$$

$$F = \{d\}$$

$$\delta =$$

	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 ."

" M ACCEPTS A ."

THE EMPTY STRING
 ϵ (epsilon, ϵ)

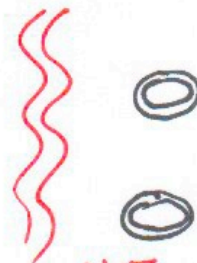
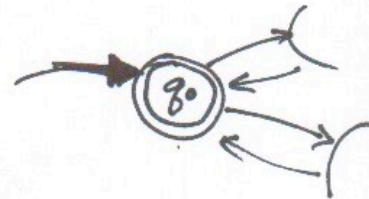
THE EMPTY LANGUAGE

$$\emptyset = \{\}$$

NOTE:

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

$$\epsilon \neq \emptyset$$



NOT REACHABLE

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

DESIGN EXAMPLE

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

Want to recognize ...

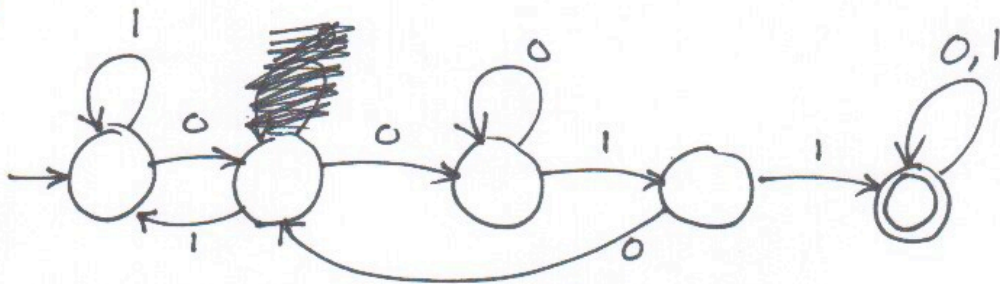
Any string that does NOT
contain 0011 in it.

How to proceed?

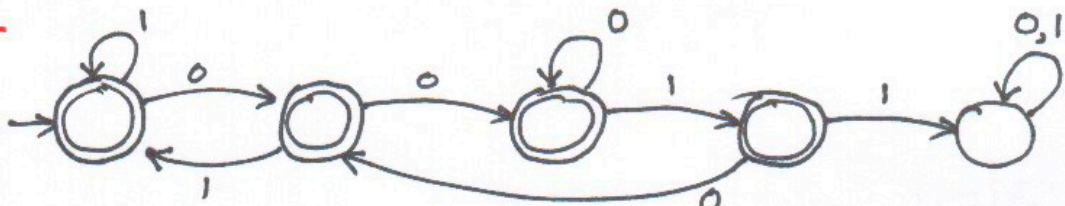
Try a simpler problem.

A string that does contain
0011 in it.

M_1



M_2



$L(M_1)$ = The Language that F.S.M.
 M_1 accepts

= The set of strings over
 $\{0,1\}^*$ that contain

$L(M_2)$ = The set of strings that
do not contain 0011 as a substring.

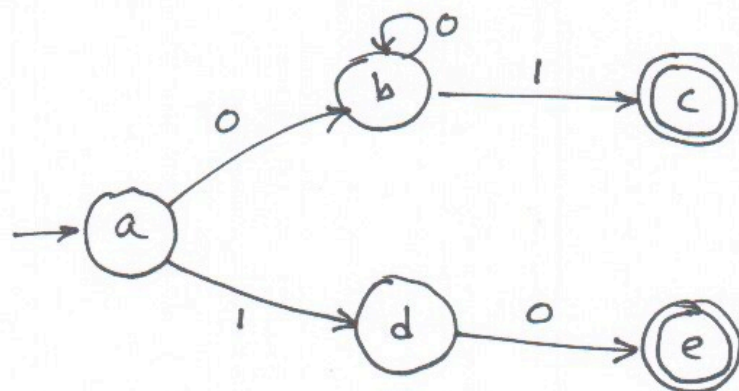
COMPLEMENTING A LANGUAGE.

(They are sets after all.)

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

Universe given by Σ ; ie. $\{0,1\}^*$

WHAT DOES THIS F.S.M. RECOGNIZE?



Recognizes 10

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

$L = \{w \mid w \text{ is either } 10 \text{ or a string of at least one } 0 \text{ followed by a single } 1\}$

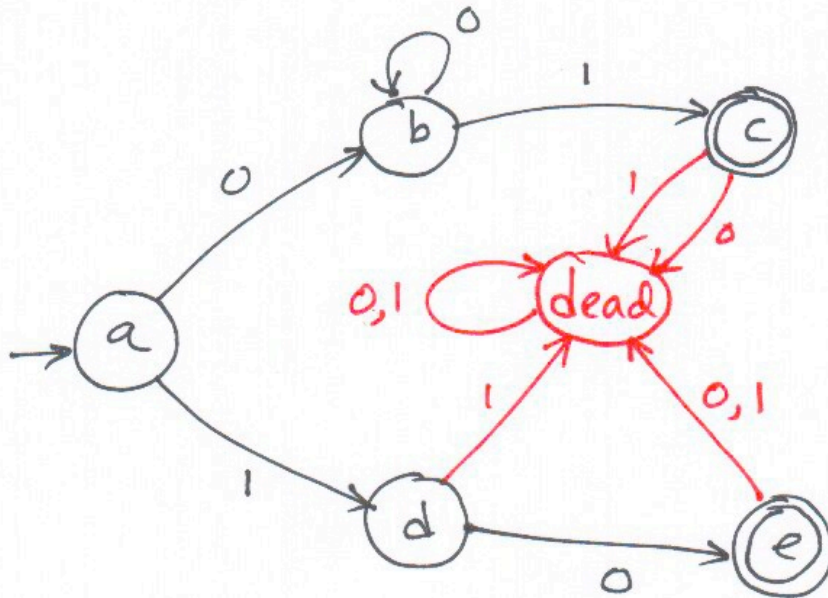
What about

111

1010

} What happens?

DEAD STATES



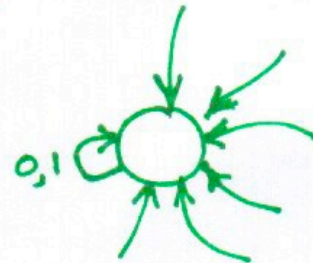
δ is a function

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

M accepts w if there is a
sequence of states

$r_0, r_1, r_2, \dots, r_N$ in Q

such that

$$r_0 = q_0$$

$$\delta(r_i, w_{i+1}) = r_{i+1} \quad \text{for } 0 \leq i \leq N-1$$

$$r_N \in F$$

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 very limited.

Cannot store the string.

Cannot "count"

Not Regular:

w^N

01101 01101
w w

$0^N 1^N$

000000 111111

Imagine a String from ⁶ here to ⁶ the Moon. You are trying to recognize it. Your only memory: A single small number (i , # of states)



...| a | b | a | c | b | c | a | a | e | g | b | a | c | ...

state = 85



DESIGNING A F.S.M.

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

Binary Numbers that are evenly divisible by 3.

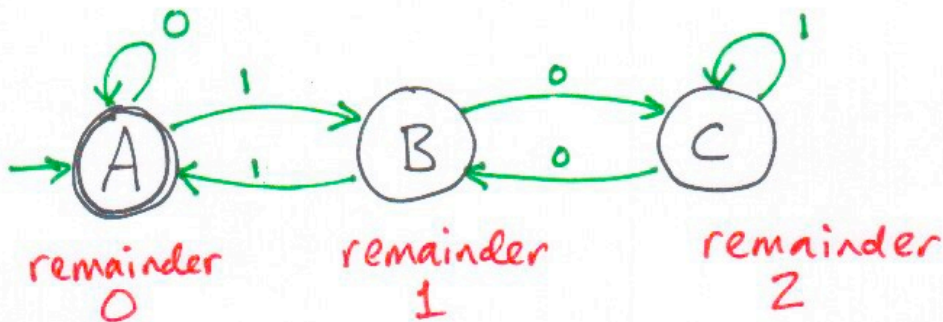
0	0
11	3 3
110	6
1001	9
1100	12
1111	15
10010	18

Idea:

State A - divisible with remainder 0

State B - divisible with remainder 1

State C - divisible with remainder 2



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

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

EXAMPLE

$$\Sigma = \{a, b, c, \dots, z\}$$

$$A = \{aa, b\}$$

$$B = \{x, yy\}$$

$$A \cup B = \{aa, b, x, yy\}$$

$$A \circ B = \{aaax, aayy, bx, byy\}$$

$$A^* = \{\epsilon, aa, b, aaaa, aab, baa, bb, aaaaaa, aaab, aabaa, aabb, \dots\}$$

THEOREM

The class of Regular Languages is CLOSED under UNION.

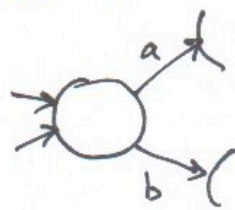
If L_1 and L_2 are regular languages, then so is $L_1 \cup L_2$.

PROOF (BY CONSTRUCTION)

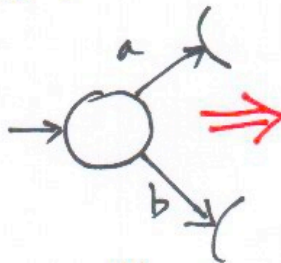
Assume $L_1 = L(M_1)$
 $L_2 = L(M_2)$

Build M to recognize $L_1 \cup L_2$.

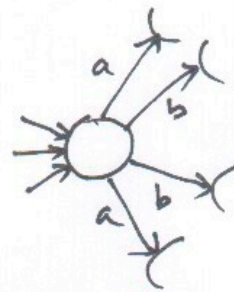
Combine machines:



M_1



M_2



M

WHOOPS!
Not a F.S.M.!

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.

$$(\text{OR: } \Sigma = \Sigma_1 \cup \Sigma_2)$$

APPROACH:

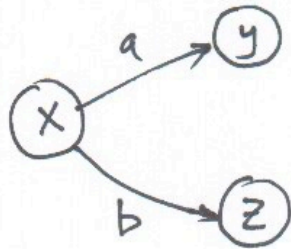
Each state in new machine represents two states

One from M_1

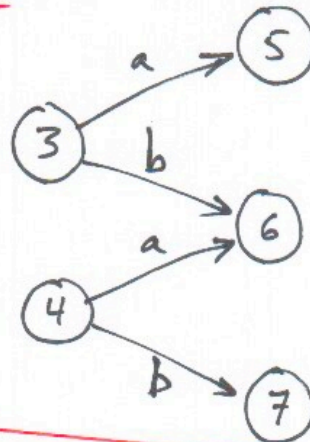
One from M_2

(Lots of possible combinations.)

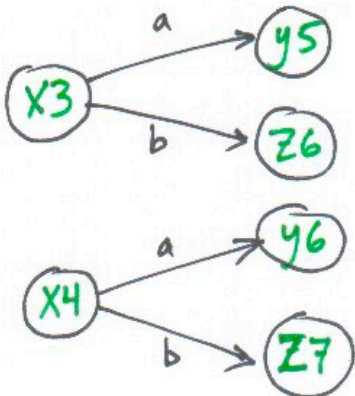
M_1



M_2



M



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

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

$$q_0 = (q_1, q_2)$$

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

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...

NON DETERMINISM

NONDETERMINISM

NFA = ~~NONDETERMINISTIC~~ = NONDETERMINISTIC FSM

UP TO NOW... DFA: Det. Finite Automaton

FSM. = Deterministic FSM, = DFSM

DETERMINISM

GIVEN THE CURRENT STATE,
WE KNOW WHAT THE NEXT
STATE IS.

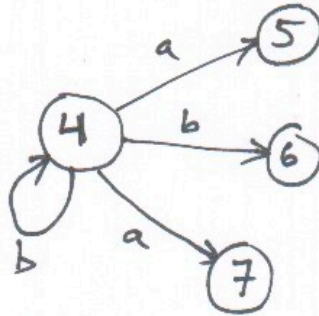
- * ONLY ONE UNIQUE NEXT STATE
- * NO CHOICES
- * NO RANDOMNESS
- * NO ORACLES
- * NO CHEATING

Normal computers are DETERMINISTIC.

(Except they take random inputs,
e.g., the timing of keystrokes
so... forget about it!)

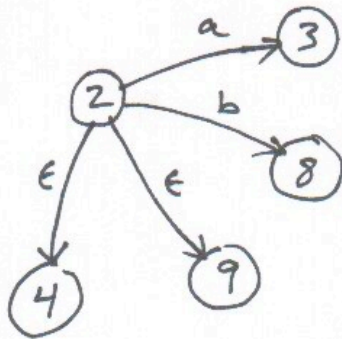
Now we will allow

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



Which edge should you take???

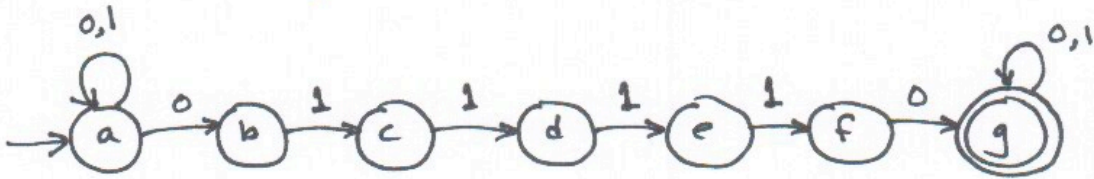
- EPSILON EDGES



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 ~~NFSM~~ accepts.

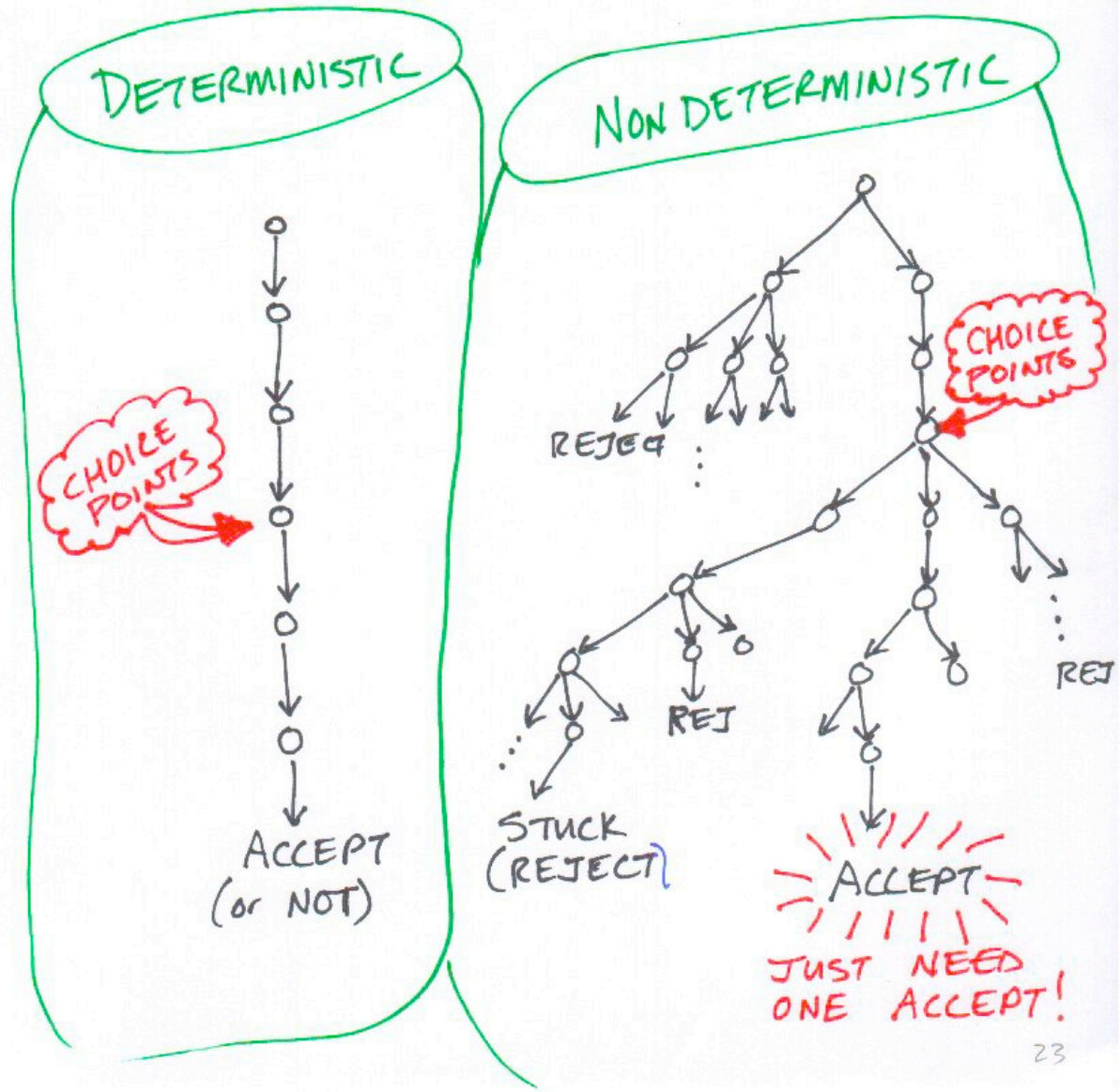
NFA

LOTS OF CHOICES - WHICH ONE TO TRY?

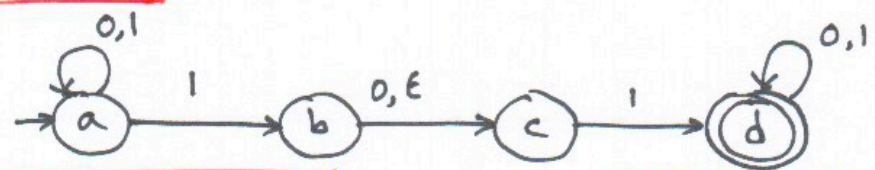
* Try them all.

* Make the right choice at each point.

EQUIVALENT!



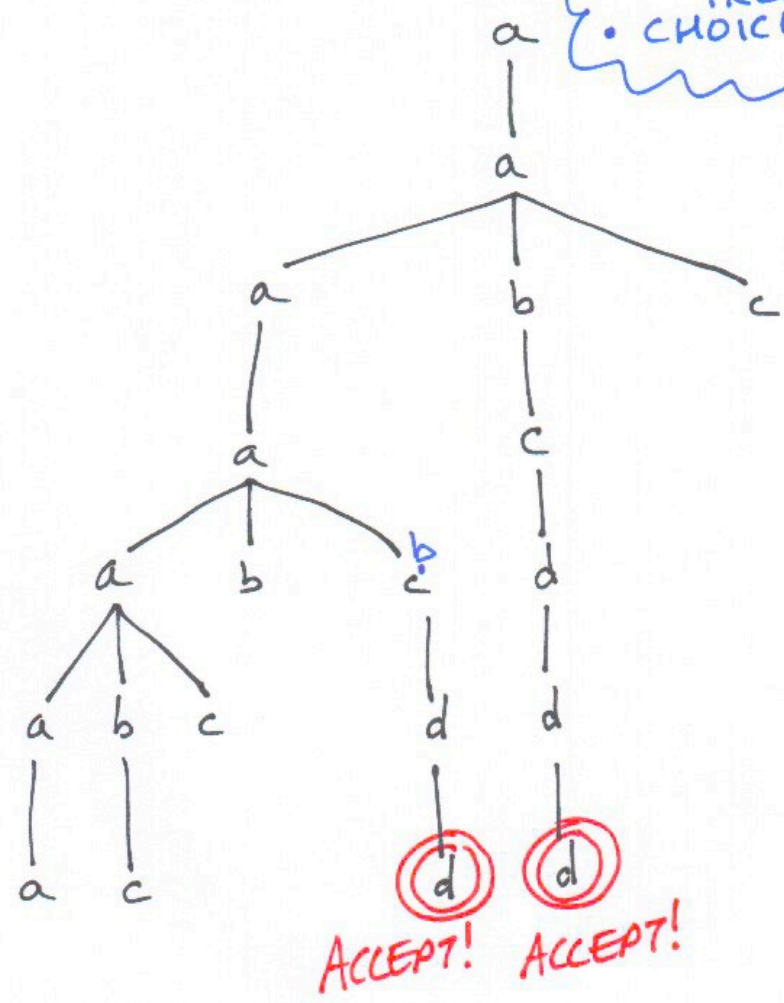
EXAMPLE



STRING: 010110

• COMPUTATION TREE
• CHOICE TREE

0
1
0
1
1
0



THEOREM

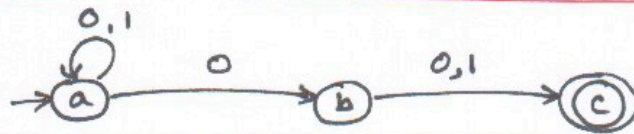
FOR EVERY
NONDETERMINISTIC F.S.M.
THERE IS AN EQUIVALENT
DETERMINISTIC F.S.M.

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

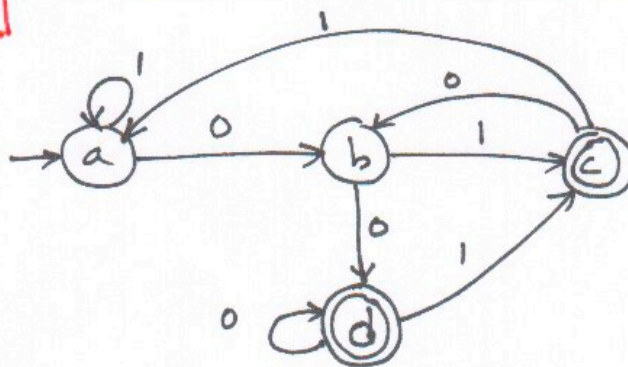
EXAMPLE

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

NFA



FSM



EXAMPLE

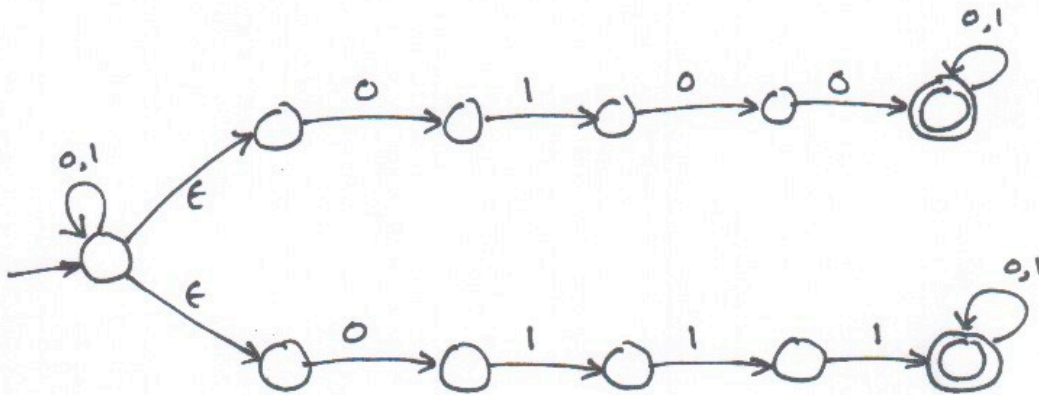
String contains either

... 0100...

or ... 0111...

When to start looking?
Which string to look for?

NONDETERMINISM!



FORMAL DEFINITION OF NONDETERMINISTIC FINITE STATE MACHINE

Σ = Alphabet

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

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

Q = states

Σ = Alphabet

q_0 = Start State, $q_0 \in Q$

F = Accept States, $F \subseteq Q$

δ = Transition Function

$P(S)$ = Powerset of S

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

BEFORE:

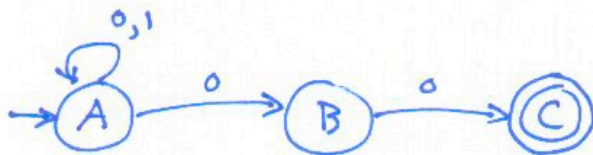
	a	b	c
q_0	q_1	q_2	q_3
q_1	q_0	q_1	q_3
	\vdots	\vdots	

NOW:

	a	b	c	ϵ
q_0	$\{q_1, q_2\}$	$\{q_2\}$	$\{q_1, q_3\}$	$\{q_1\}$
q_1	$\{\dots\}$	$\{\dots\}$	$\{\dots\}$	$\{\dots\}$
	\vdots	\vdots	\vdots	\vdots

EXAMPLE

ACCEPT ALL STRINGS OVER $\{0,1\}^*$
ENDING WITH "00".



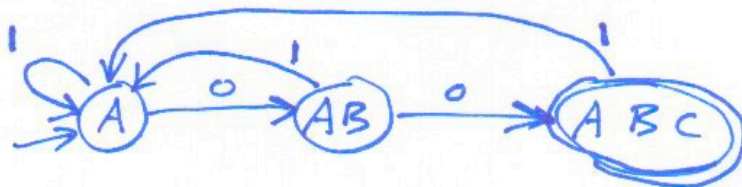
PRETEND TO SIMULATE. USE A FINGER
FOR EACH STATE WE COULD
BE IN.

00100

A AB ABC B BC C

AC \emptyset

$N = \#$ of states in NFA
Worst case $\#$ of states in DFBA?
 2^N



THEOREM

EVERY NONDETERMINISTIC FSM. HAS AN EQUIVALENT DETERMINISTIC F.S.M.

("EQUIVALENT" = Recognizes the same language.)

PROOF BY CONSTRUCTION

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

NONDETERMINISTIC
FSM

Assume k states.

DFA will have 2^k states.

CONSTRUCT ~~THE~~ DFA:

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

$$Q' = P(Q)$$

$$\delta'(R, a) =$$

$$= \left\{ g \mid g \in Q \text{ and } g \in \delta(r, a) \text{ for some } r \in R \right\}$$

OF EQUIVALENTLY:

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

$R =$ a state in Q'
ie. in the DFA
 $a =$ an input symbol

$$q_0' = \{q_0\}$$

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

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

AC

BC

ABC

C

But What about ϵ -edges?

Consider a state in the DFA
we're constructing?

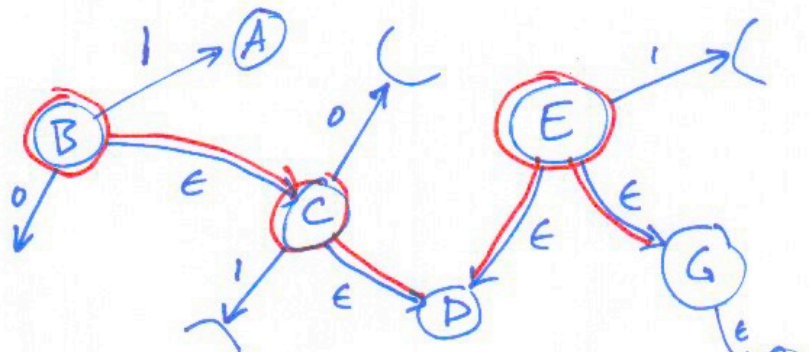
BCE

A state "R"
in DFA

Look back at the ~~NFA~~ NFA.

What states can we reach by
going thru ϵ -edges?

Also include ~~B~~ B, C, and E.



DEFINE

$E(R) = \{g \in Q \mid g \text{ can be reached from a state in } R \text{ by going along 0 or more } \epsilon\text{-edges.}\}$

"EPSILON-CLOSURE"

= BCEGH

Modify the transition function:

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

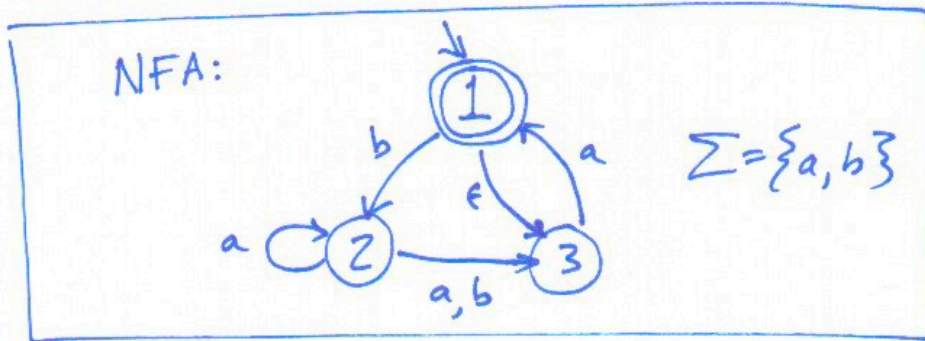
for some $r \in R$

Also, modify the start state in the constructed DFA:

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

END OF PROOF

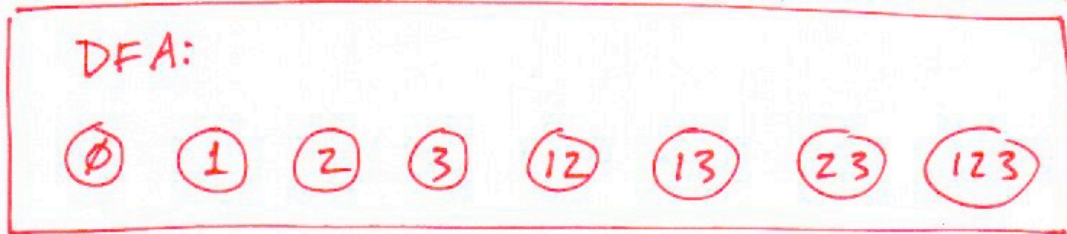
EXAMPLE: CONVERT AN NFA TO A DFA.



What states in the DFA?

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

RED



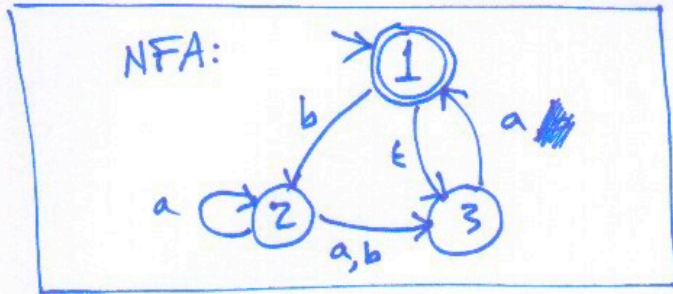
Start state?

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

$$E(1) = 13$$

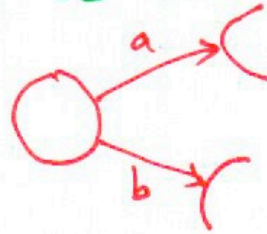
ACCEPT STATES? ANY THAT CONTAIN 1

$$F' = 1 \quad 12 \quad 13 \quad 123$$



TRANSITION FUNCTION?

FOR EACH STATE, WE NEED TWO ~~EDGES~~ EDGES:



$$\delta'(\textcircled{1}, a) = \textcircled{\emptyset}$$

Since no edges labeled "a" leave $\textcircled{1}$.

$$\delta'(\textcircled{1}, b) = \textcircled{2}$$

From $\textcircled{1}$ we can get to $\textcircled{2}$, but no further with ϵ -edges

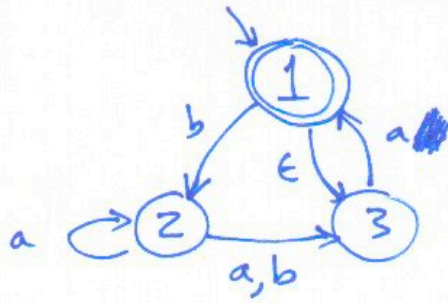
$$\delta'(\textcircled{2}, a) = \textcircled{23}$$

$$\delta'(\textcircled{2}, b) = \textcircled{3}$$

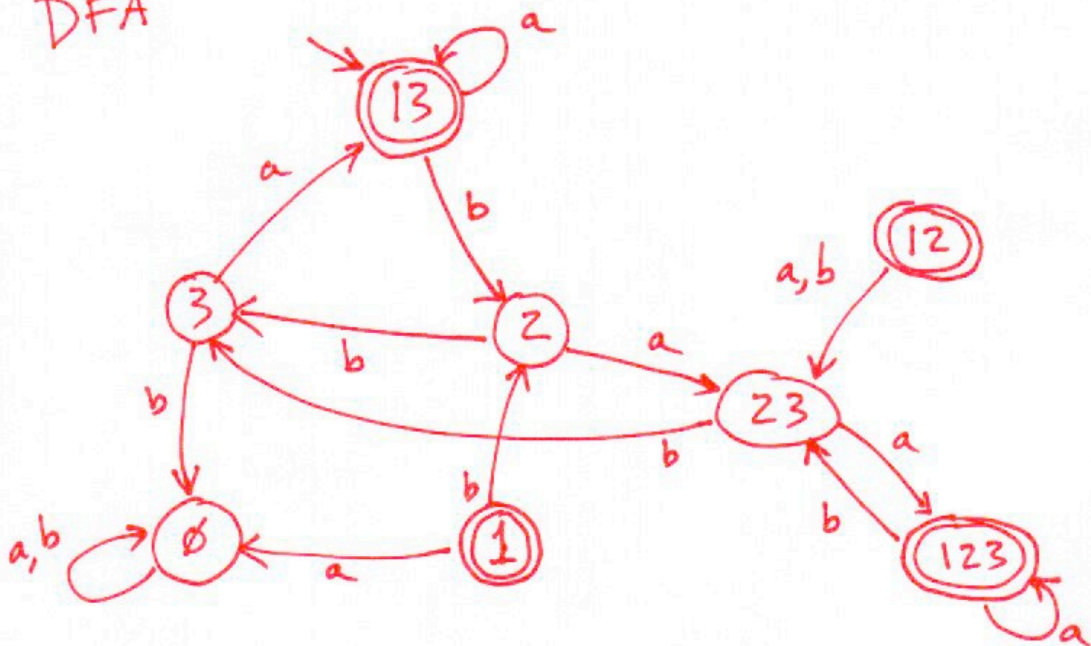
$$\delta'(\textcircled{3}, a) = \textcircled{13}$$

Can get to $\textcircled{1}$ but can get to $\textcircled{3}$ by following ϵ -edge.

NFA:



DFA



NOTE: ① AND ⑫ ARE UNREACHABLE.
THEY CAN BE REMOVED.

THEOREM

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

"CLOSURE" OF A LANGUAGE.

PROOF

ASSUME A_1 AND A_2 ARE
REGULAR LANGUAGES.

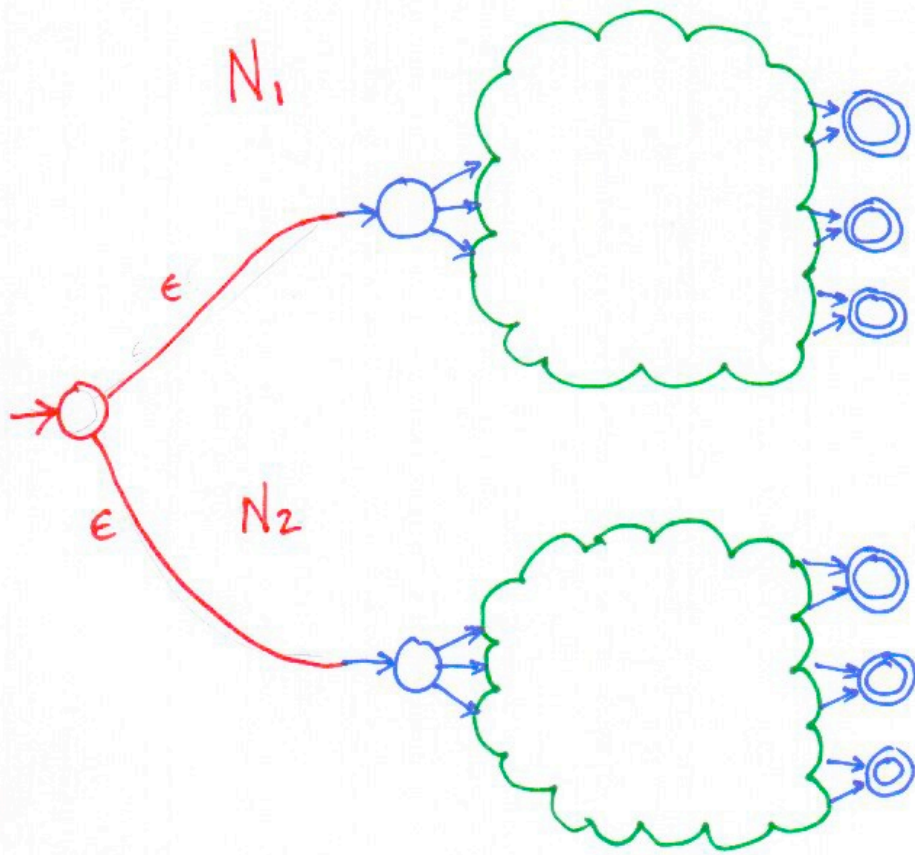
SHOW $A_1 \cup A_2$ IS REGULAR.

ASSUME NFA N_1 RECOGNIZES A_1

NFA N_2 RECOGNIZES A_2

COMBINE THEM;

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



FORMALLY:

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

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

CONSTRUCT

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

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

q_0 is new start state

$$F = F_1 \cup F_2$$

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

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 $y \in A_2$.

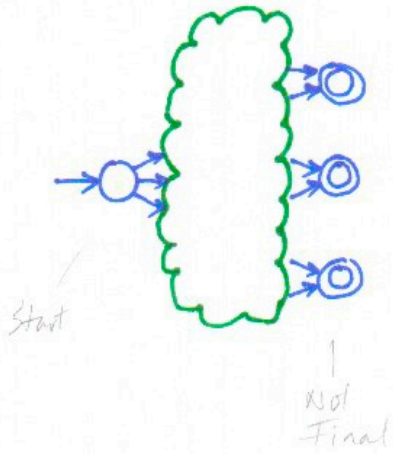
PROOF

SAME APPROACH.

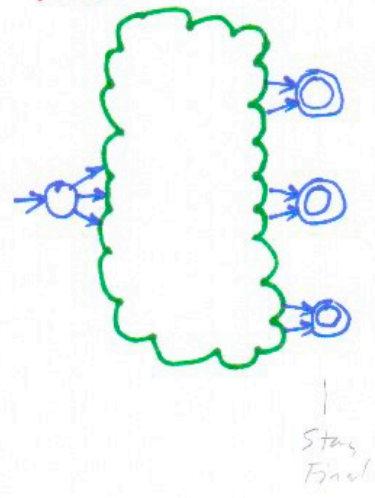
Assume NFA N_1 recognizes A_1
and NFA N_2 recognizes A_2

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

N_1



N_2



ϵ

FORMALLY

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

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

Construct

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

$$Q = Q_1 \cup Q_2$$

$$q_0 = q_1 \quad \text{the start state of } N_1$$

$$F = F_2 \quad \text{the final states of } N_2$$

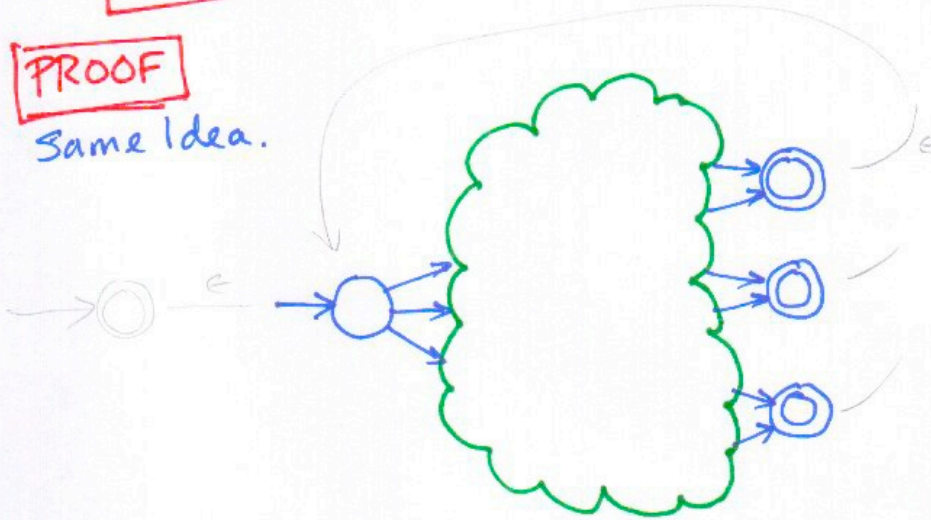
$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{if } q \in Q_1 \\ \delta_2(q, a) & \text{if } q \in Q_2 \\ \delta_1(q, a) \cup \{q_2\} & \text{if } q \in F_1 \text{ and } a = \epsilon \\ \delta_1(q, a) & \text{if } q \in F_1 \text{ and } a \neq \epsilon \end{cases}$$

THEOREM

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

PROOF

Same Idea.



NEXT TIME

Regular Expressions

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

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

Regular Expressions

//

Regular Languages

//

Finite State Machines

// //

DFA = NFA

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

ϵ is a Reg. Expression

\emptyset is a Reg. Expression

(R_1) is a Reg. Expression

Star binds tightest.

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

Concatenation binds tighter than Union.

$$ab \cup c = (ab) \cup c \\ \neq a(b \cup c)$$

Other notations

$$ab/c = (ab)/c \\ \neq a(b/c)$$

$$a^* = \{a\} = \{a\}^*$$

"One or more"

$$a^+ = aa^* = \{a\}^+$$

"Optional"

$$[a] = a/\epsilon = (a \cup \epsilon) = a^?$$

Parsing Practice

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

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

$$aab | caab | caa$$
$$= (aab) | (caab) | (caa)$$

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

$$d | ab^* | cd^* = ?$$

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

$$= d | (a(b^*)c(d^*))$$

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

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

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

$\{abc, ac\}$

\emptyset

$\{\}$

$a(b \cup c)\emptyset$

$\{\}$

\emptyset^*

$\{\epsilon\}$

REGULAR EXPRESSION

Each Regular Expression describes a Language. Which Language?

$$L(R) = ?$$

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

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

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

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

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

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

$$L((R_i)) = L(R_i)$$

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

\Rightarrow Regular Expressions
describe Regular Languages.

THEOREM

A LANGUAGE IS REGULAR IFF
SOME REGULAR EXPRESSION
DESCRIBES IT.

LEMMA 1:

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

PROOF #1: Use closure of $\cup \circ *$.

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

LEMMA 2:

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

PROOF: Start with a DFA
Build a GENERALIZE NONDETERMINISTIC
FINITE STATE AUTOMATON (GNFA).
Reduce it, yielding a Reg. Expression 48

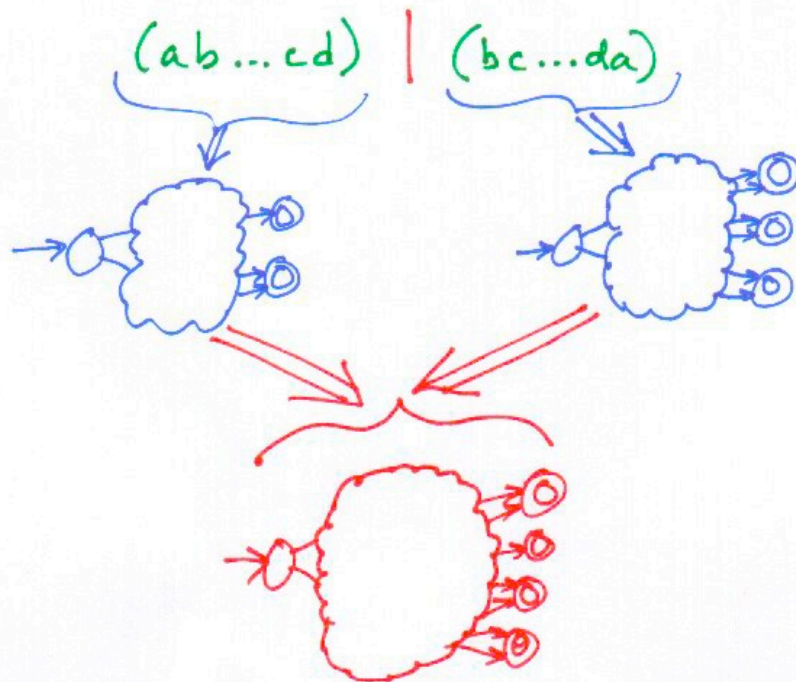
LEMMA 1

- 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)



DEFINITION OF REGULAR EXPRESSIONS

$$R = a$$

$$R = R_1 | R_2$$

$$R = R_1 \circ R_2$$

$$R = R_1^*$$

$$R = \epsilon$$

$$R = \emptyset$$

$$R = a$$



$$R = \epsilon$$



$$R = \emptyset$$



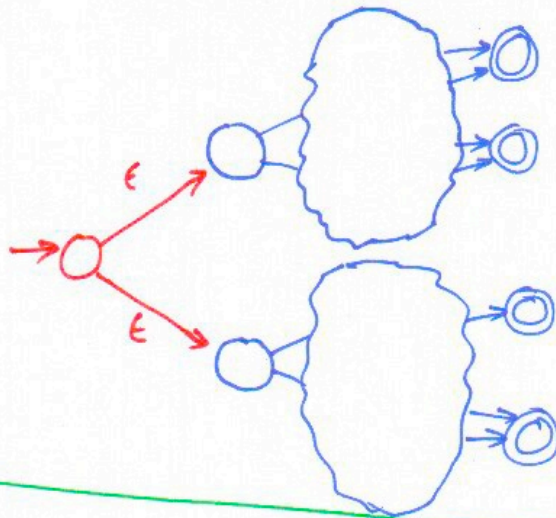
$$R = R_1 | R_2$$

$$R = R_1 \circ R_2$$

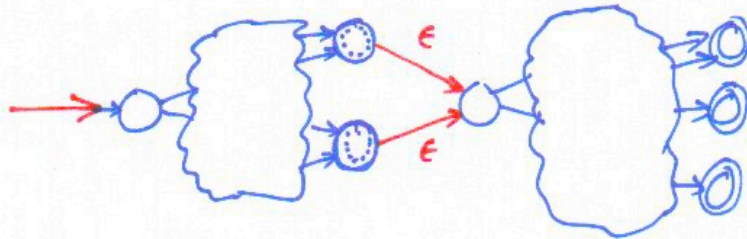
$$R = R_1^*$$

Same construction
used in proof
of closure.

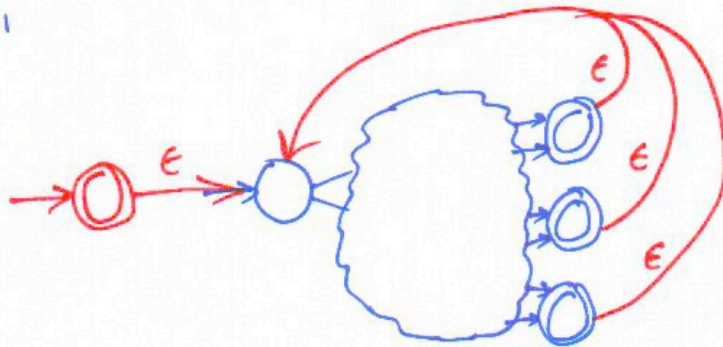
R_1/R_2



$R_1 \circ R_2$



R_1^*



LEMMA 2

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

GENERALIZED NONDETERMINISTIC FINITE AUTOMATON (GNFA)

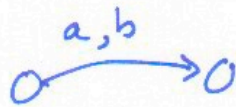
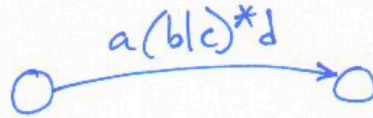
LIKE A N.F.A.

EXCEPT:

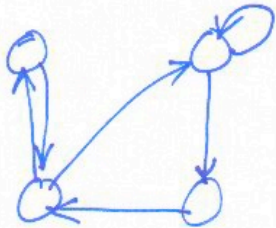
- ⊗ EDGES ARE LABELED
WITH REGULAR EXPRESSIONS!
- ⊗ ONLY ONE ACCEPT STATE
- ⊗ THERE IS EXACTLY ONE
EDGE FROM EVERY STATE
TO EVERY OTHER STATE.
(INCLUDING AN EDGE TO SAME STATE)
- ⊗ EXCEPT: NO EDGES GOING TO THE
START STATE.
- ⊗ EXCEPT: NO EDGES GOING OUT
OF THE ACCEPT STATE.

NFA

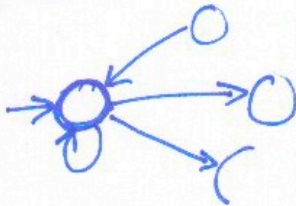
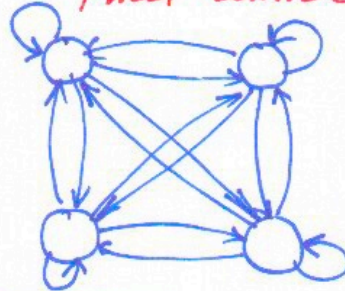
GNFA



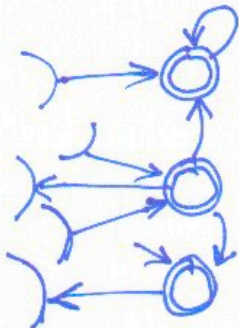
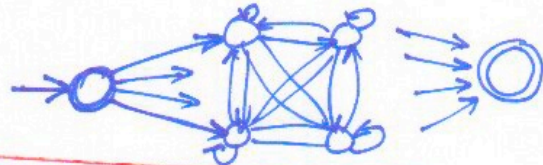
ONLY ONE EDGE BETWEEN STATES



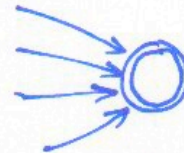
FULLY CONNECTED

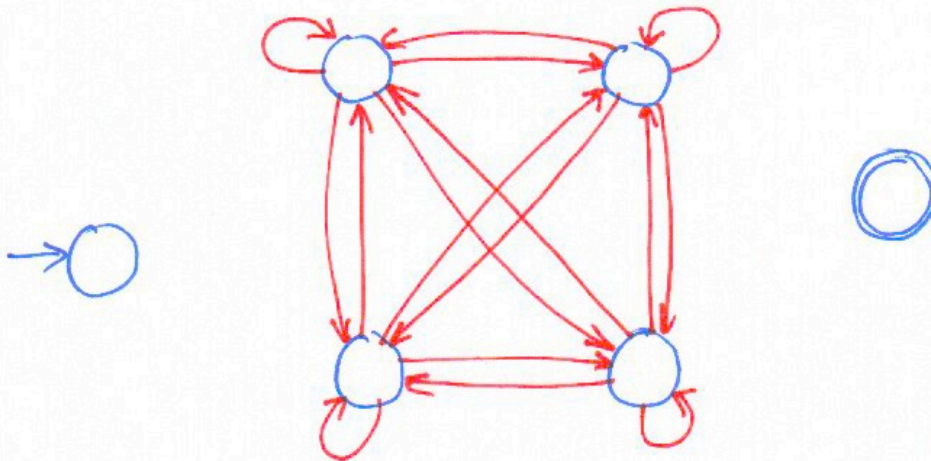
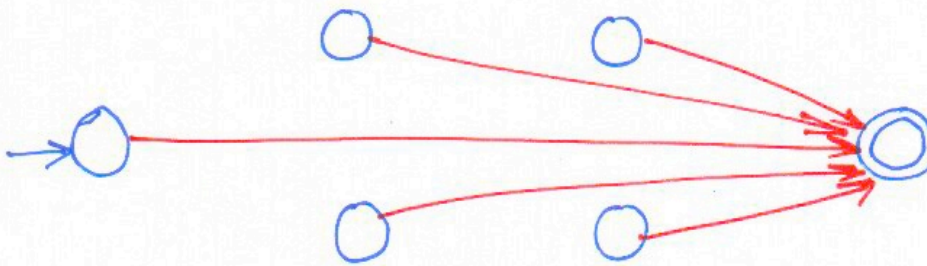
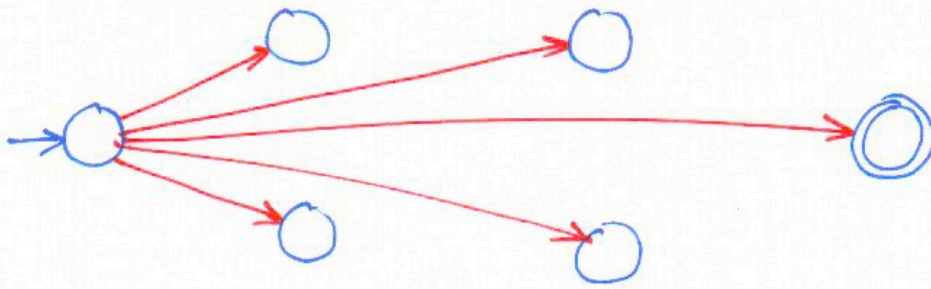


NO EDGES TO START STATE



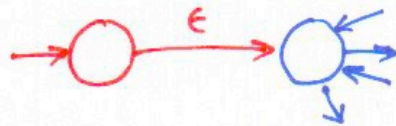
ONLY ONE FINAL STATE;
NO EDGES OUT OF IT.



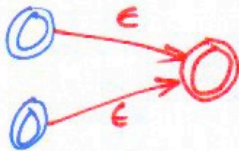


STEP 1 CONVERT FROM DFA TO GNFA.

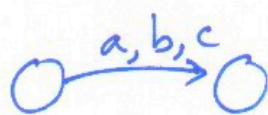
Add a start state.



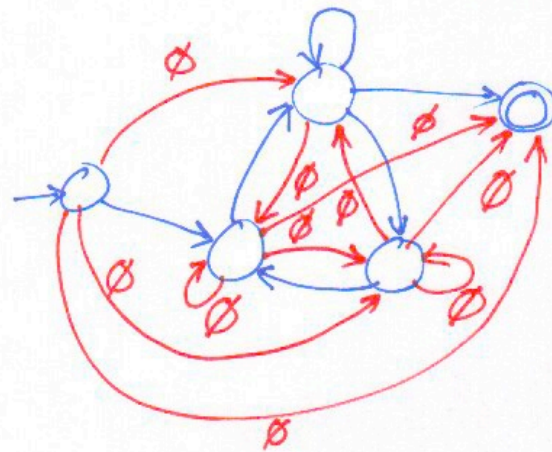
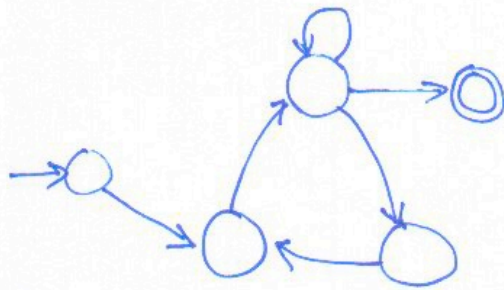
Add a new ACCEPT STATE.



Eliminate multiple edges with UNION.



Add missing edges (with \emptyset).



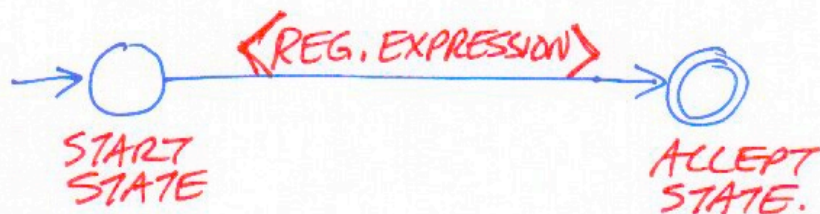
STEP 2:

CHOOSE A STATE.

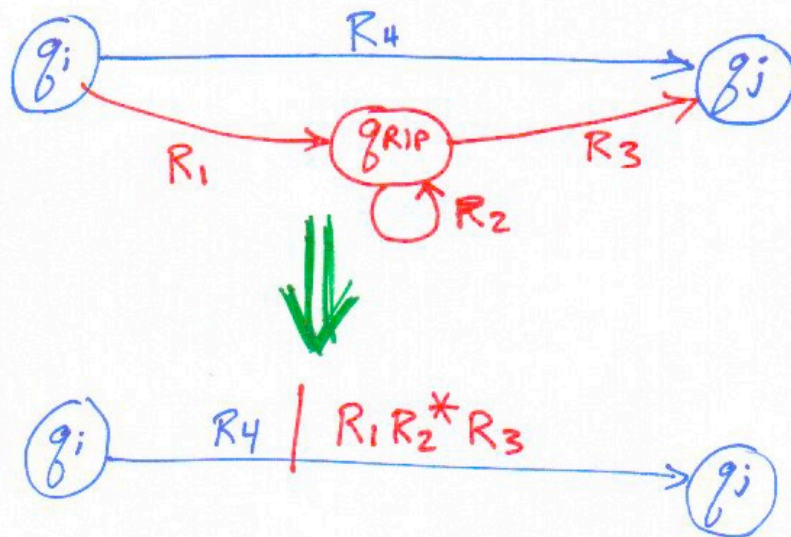
REMOVE IT.

MODIFY THE MACHINE SO
IT STILL ACCEPTS THE
SAME LANGUAGE.

REPEAT UNTIL THERE
ARE ONLY 2 STATES LEFT.



- Select a state at random.
(Don't select the START or ACCEPT states.)
- Call it q_{RIP} .
- Rip this state out.
- Remove q_{RIP} and all edges to/from it.
- Modify all other edges.
So that the resulting machine still accepts same language.



EXAMPLE

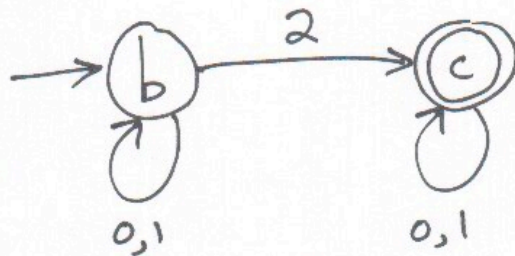
Will want to simplify
our Regular Expressions.

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

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

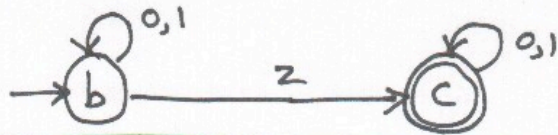
$$\emptyset | R = \{\} \cup R = R$$

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

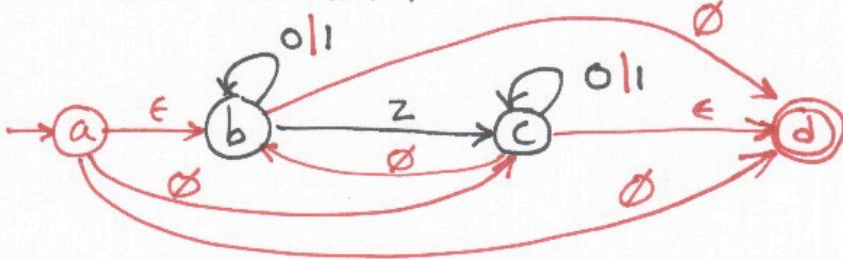


DESIRED ANSWER:

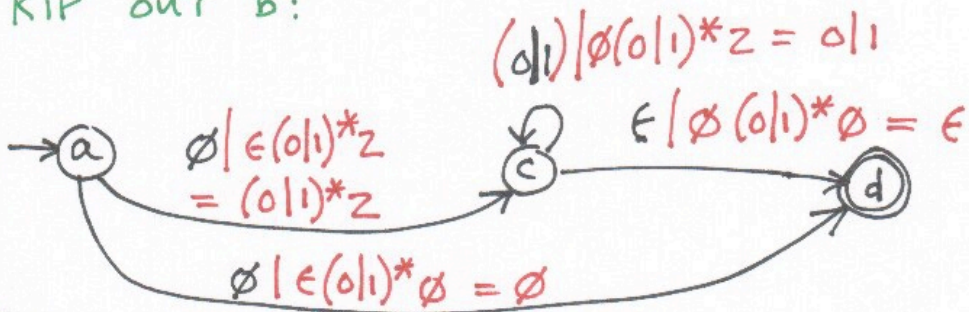
$$(0/1)^* 2 (0/1)^*$$



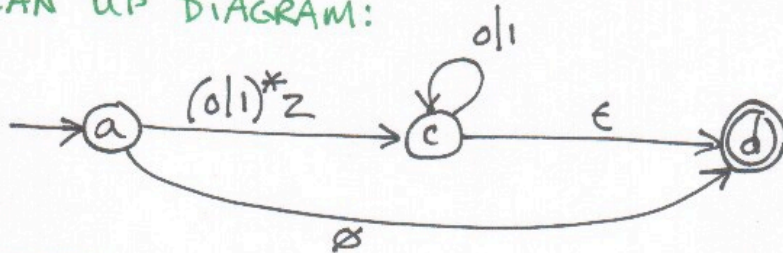
CONVERT TO GNFA:



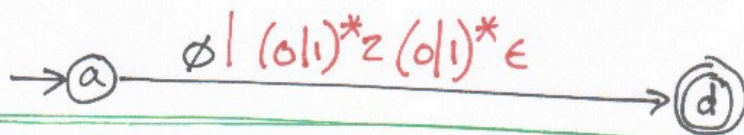
RIP OUT b:



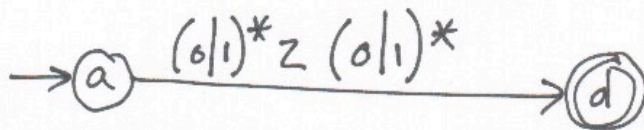
CLEAN UP DIAGRAM:



RIP OUT c:



SIMPLIFY:



NON-REGULAR LANGUAGES

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

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

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

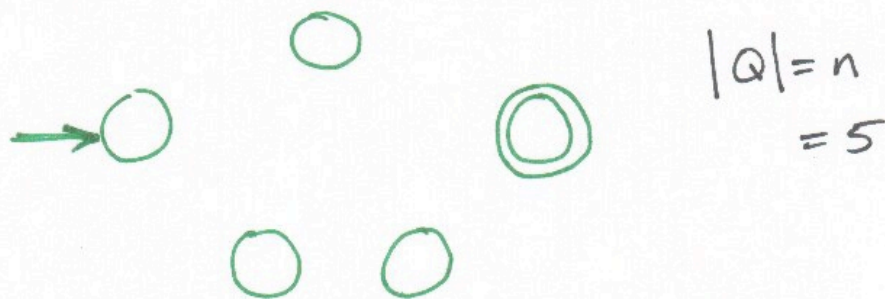
↖ SURPRISE! This is regular.

HOW CAN WE PROVE A LANGUAGE
IS NOT REGULAR?

THE PUMPING LEMMA!

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

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



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:

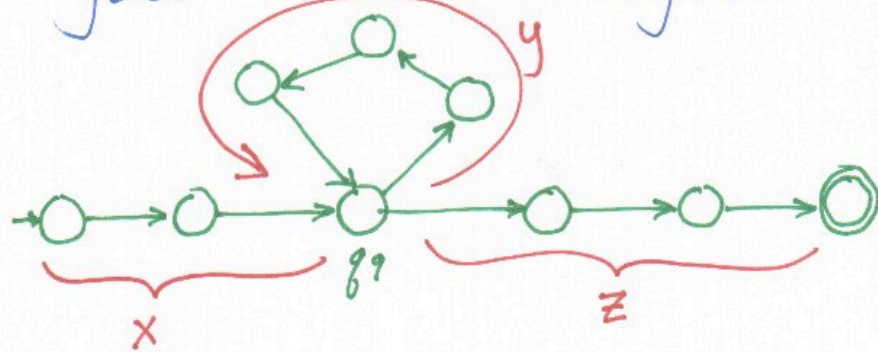
"If you can go around the cycle once, then you can go around it two times, and that string MUST ALSO BE IN the language."

(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.)

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



String $s = xyz$

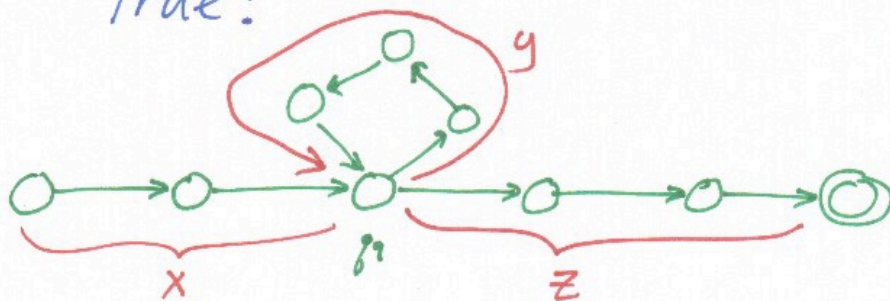
But Note: All strings of the form

$$s = xy^i z$$

are in the language.

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

What can we say must be true?



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

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

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.

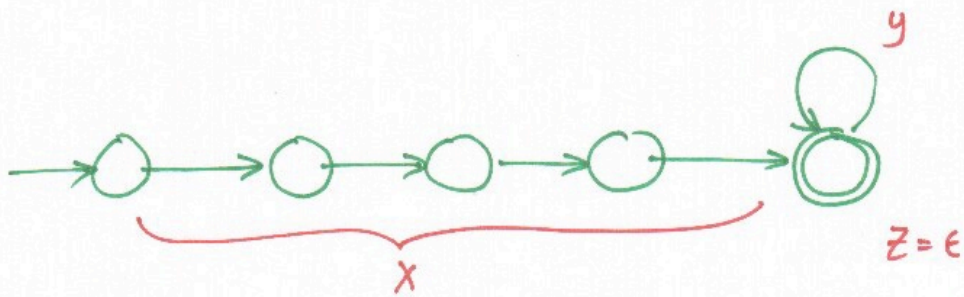
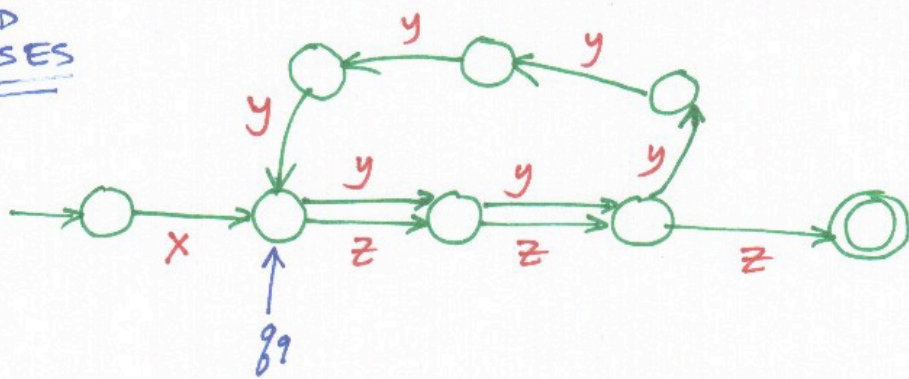
$|xy| \leq P$ ← You have to hit the cycle before the string gets longer than P .

The PUMPING LENGTH.

Depends on the language, not any specific F.S.M.

Long enough that ~~some~~ ~~some~~ FSM will have a cycle.

ODD
CASES



$|y| > 0$ just barely

$|xy| \leq p$ just barely.

$p = 5$

NOTE:

We are saying:

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.

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

$$S = xyz$$

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

such that all these conditions hold:

CONDITION 1

$$xy^iz \in A \quad \text{for every } i \geq 0$$

CONDITION 2

$$|y| > 0$$

CONDITION 3

$$|xy| \leq p$$

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 ALL 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 \Rightarrow CONTRADICTION!

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

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

Let's 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

$$0^7 1^7 = \underbrace{0000000}_y \underbrace{0111111}_y \underbrace{11111}_y$$

CASE 1: 00000 11111
 y

So xy^iz must be in B .

$$xy^2z = 00000000011111$$

CASE 2: 00000 11111
 y

$$xy^2z = 00000111111111$$

CASE 3: 00000 11111
 y

$$xy^2z = 00000110011111$$

Not of the form 0^n1^n

ALSO RECALL CONDITION 3

$$|xy| \leq p$$

$$S = 0^p 1^p = \overbrace{00000}^p \cdot \overbrace{11111}^p$$

x y →

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

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

$$S = x y z$$

$$\text{condition 2: } |y| > 0$$

$$\text{condition 3: } |xy| \leq p$$

$$p=7: 0^7 1 0^7 1 = 0000000 \underbrace{000}_{y} 100000001$$

- So y must be all zeros.
- So xy^2z is not in F .
- Contradiction!

F is not regular.

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!

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