classmate
Date 12/7/17
Page

Formal language and Automata Theory

1. An introduction to formal languages & automata theory — Peter Linz
   — Narosa Pub 2nd or 3rd.

2. Introduction to automata theory, languages & computation 3rd Edition.
   John E Hopcroft
   Rajeev Motwani — Lecture slide | Video Lecture.
   Jeffrey D. Ullman
   Pearson Education

3. Introduction to theory of computation.
   Michael Sipser.

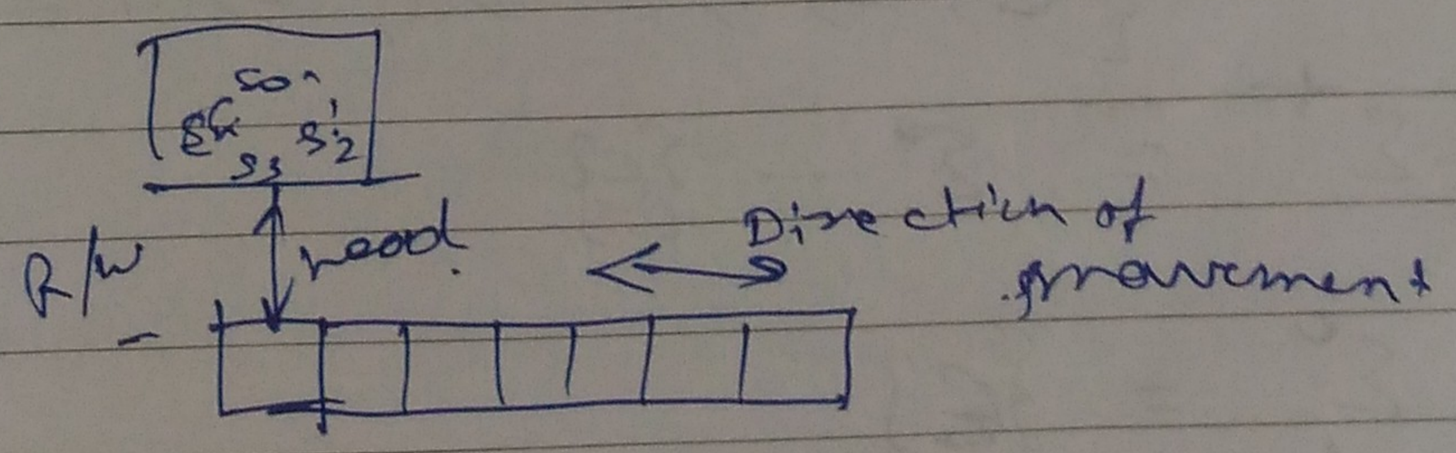
Halting Problem.

Algo i, input i → [ALGO] → Algo halts with input i ?
Finite control
Y/N.

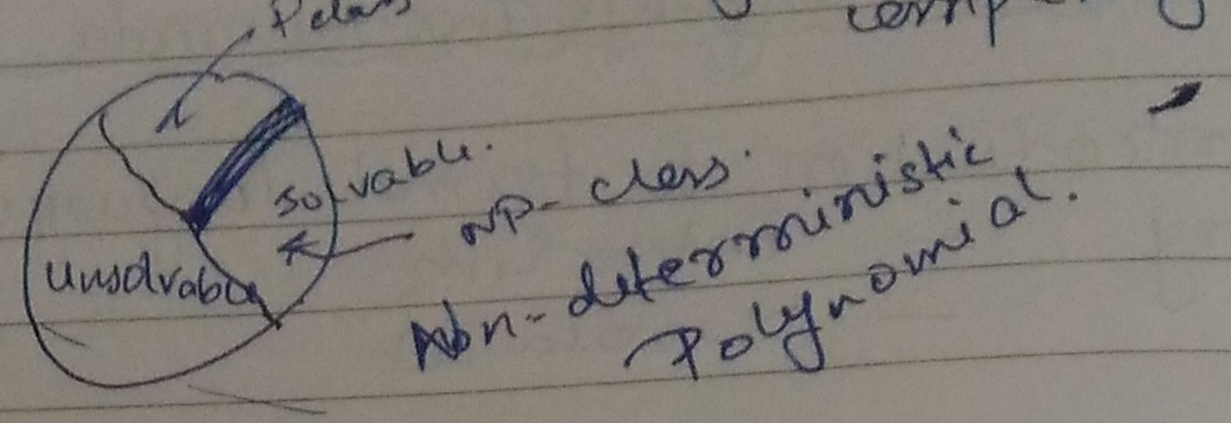
Can't be solved.

All problems
solvable unsolvable.

Turing Machine.



Finite automata.
Push down automata.



Alphabet (Σ)

Any finite set of symbols, eg. $\Sigma = \{0, 1\}$
or $\Sigma = \{a, b, c\}$ etc

Strings

- A finite sequence of symbols chosen from some alphabet.
- ϵ . stands for the empty string. (string of length 0)
- Star closure of alphabet Σ^*
- Σ^* denotes the set of all strings formed with the symbols taken from Σ

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

0 is a string
1 " " " "
00 " " " "
01 " " " "
10 " " " "
11 " " " "

$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11 \ldots\}$

$\Sigma^+ = \Sigma^* - \{\epsilon\}$

$\Sigma^0 = \{\epsilon\}$

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

$\Sigma^2 = \{00, 01, 10, 11\}$ length of all string $= 2$

- **Language (L)**

  a subset of Σ^* for some alphabet Σ

- Eg: The set of strings of 0's and 1's with no two consecutive 1's.

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

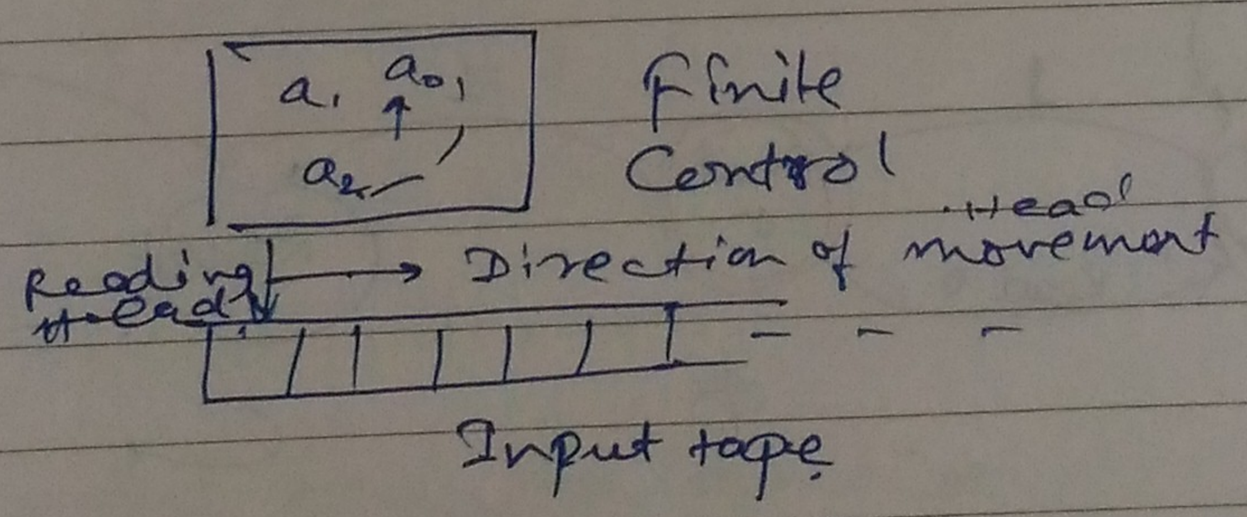
  $L = \{\epsilon, 1, 0, 10, 0, 00, 000, 001, 010, 101, 100$

* Finite Automata : Informal Explanation

- Finite automata are finite collections of states with transition rules that take you from one state to another

- Original application was sequential switching circuits where the "state" was the settings of internal bits

- Today, several kinds of software can be modeled by Finite Automata



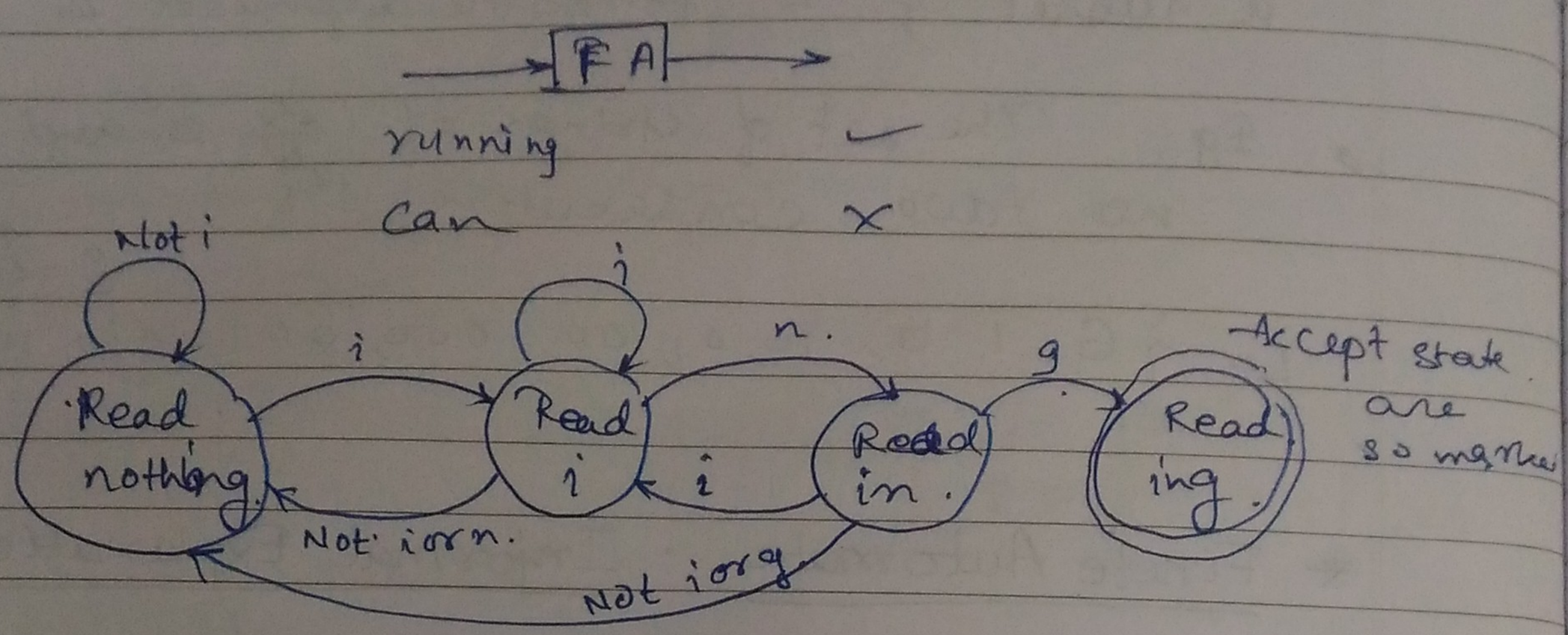
* Representation of Finite Automata.

- Simplest representation is often a graph.
  - Nodes = States.
  - Arcs indicate state transitions.
  - Labels on arcs tell what causes the transition.

Date 19/7/17

Eg:

Recognizing strings Ending in "ing"

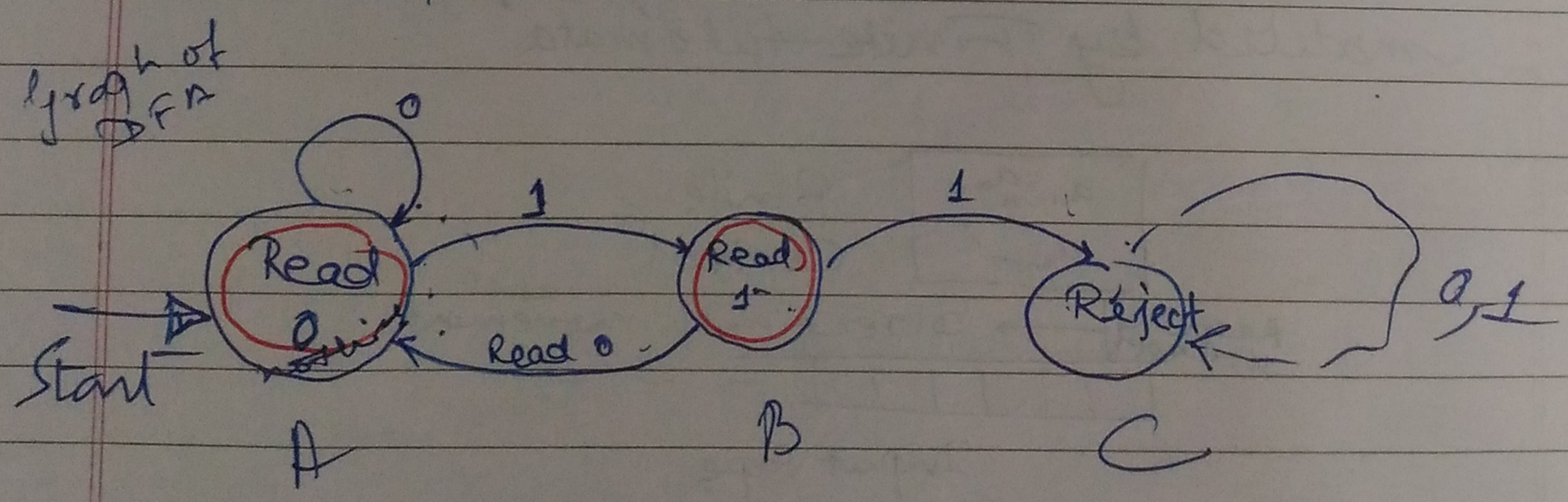


Example:

Design a DFA that accepts all binary strings without two consecutive 1's

{ 0, 1, 00, 01, 10, 101 ..... } ✓

{ 0✗, 1✗ }



Deterministic Finite Automata (DFA)

A formation for defining languages consists of
1. A finite set of states (Q, typically)
2. An input alphabet (Σ, typically)
3. A transition function (δ, typically)

It takes as arguments a state and an input symbol and returns a state

For eg, $\delta(q, a)$ = the state that the DFA goes to when it is in state q and input a is recieved.

4. A start state (q_0 in Q, typically)
5. A set of final/accept states ($F \subseteq Q$, typically)

In proofs, we often talk about a DFA in "five tuple" ~~notatati~~ notation.

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

An alternative representation transition table

state \ input	0	1
A^*	A	B
B^*	A	C
C	C	C

✓ **Extended Transition Function**

✓ We describe the effect of a string of input symbols on a DFA by extending δ to a state and a string.

✓ **Induction on length of string**

Basis : $\delta(q, \epsilon) = q$

Induction : $\delta(q, wa) = \delta(\delta(q, w), a)$

w = a string.
x = an input symbol.

<u>Example</u>: Extended Delta ($\hat{\delta}$)

$\delta(B, \overset{ABC}{011}) = \delta(\delta(B, 01), 1)$
$= \delta(\delta(\delta(B, 0), 1), 1)$

now in state B.

$$= \underbrace{\delta(\underbrace{\delta(\underbrace{\delta(\delta(\underbrace{B}_{B}, 0), 1)}_{A}, 1)}_{B}, 1)}_{C}$$

Language of DFA

- Automata of all kinds define languages.
- If A is an automata L(A) is its language
- For DFA, A, L(A) is the set of strings labelling paths from the start state to a final state.
- Formally L(A) = the set of strings such that $\delta(q_0, w)$ is inf

* <u>The language of an example DFA</u> ~~can be of defined~~

{w : w is in {0, 1}* and w does not have two consecutive 1's}

- <u>Proof of set Equivalence</u>

  - Often we need to prove that two descriptions of sets are in fact the same set

✓ Here one set is "the language of the DFA", i.e, L, and the other is the set of strings of 0's and 1's with no consecutive 1's, i.e. S (say).

✓ <u>Proof</u>:

✓ In general, to prove $L = S$, we need to prove two parts:

1. $L \subseteq S$. If w is in L then w is in S
2. $S \subseteq L$ If w is in S then w is in L

* Here, one set, i.e. L "is the language of this DFA". And the other i.e S is "the set of strings of 0^s and 1^s with no consecutive 1's"

Part 1: $L \subseteq S$

✓ For the DFA we have designed

1. If $\delta(A, w) = A$ then w has no consecutive 1's and does not end in a 1.

2. If $\delta(A, w) = B$ then w has no consecutive 1's and ends in a single 1

<u>Basis</u>: $|w| = 0$, i.e. $w = \epsilon$

- (1) holds since ϵ has no 1st at all
- (2) holds since $\delta(A, \epsilon)$ is not B

$P \Rightarrow Q \quad = \quad \sim P \vee Q$

P	Q.	$P \Rightarrow Q$.
T	F	F
T	T	T
F	T	T
F	F	T

Induction

Assume (1) and (2) are true for strings shorter than w, where $|w|$ is at least 1.

Let $w = x\,a$ ← an input symbol.

IH is true for x.

Induction Hypothesis

* Let us prove (1) for $w = xa$.

$$\delta(A, w) = \delta(A, xa) = \delta(\underbrace{\delta(A, x)}_{A \text{ or } B}, a)$$

So a must be 0 (look the DFA)

By the IH x has no 11's.

Thus w has no 11's and doesnot end in a 1

* Let us prove (2) for $w = xa$

$$\delta(A, w) = \delta(A, xa)$$
$$= \delta(\underbrace{\delta(A, x)}_{A}, a)$$

a must be 1 (look the DFA)

- By IH, x has not 11's and does not end in a 1
- Thus w has no 11's and end in a 1

Date 26/7/17

Now ω we prove reverse.

Part 2: $S \subseteq L$

Contrapositive law.

$P \Rightarrow Q$

$\equiv \sim Q \Rightarrow \sim P$

Now we must prove,

If ω has no 11's, then ω is accepted by DFA

Contrapositive

If ω is not accepted by DFA then ω has 11

- The only way ω is not accepted is if it gets to state C.
- $\delta(A, w) = C$ ~~happ~~ happens if $w = x \, 1 \, y$ for B to C

  x gets to B, and y is the tail of w that follows what gets to C for the first time

- If $\delta(A, x) = B$ then surely ~~x = z~~ $x = z1$ for A to B

  some z.

  Thus $w = \boxed{z 11 y}$ and has 11.

Regular Languages

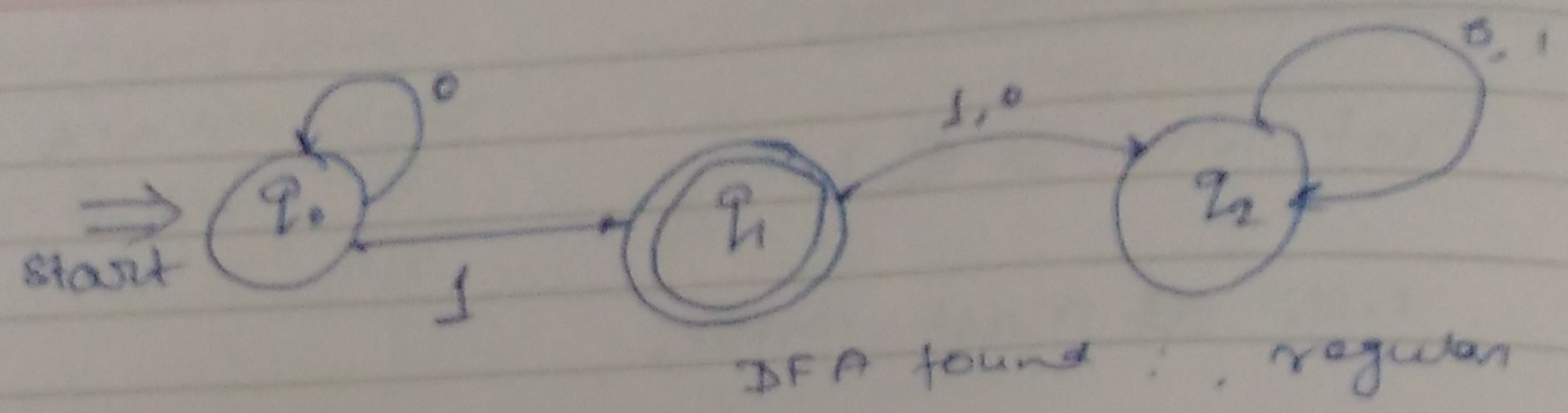
A Language L is regular if it is the language accepted by some DFA

Notice: the DFA must accept only the strings in L, no others

eg: $L = \{ a^n b \mid n \geq 0 \}$

Let $a = 0$, $b = 1$

$L = \{1, 01, 001, 0001, \ldots\}$



DFA found ∴ regular

Example: ~~of~~

A non regular Language

Let $L = \{0^n, 1^n \mid n \geq 1\}$

$L = \{01, 0011, 000111, \ldots\}$

Finite Automata cannot count

So DFA can't be designed to accept their particular languages.

i.e. no. of 0's = no. of 1's

* Intuitively regular languages "cannot count" to arbitarily high integers

Another. Example:

$L = \{\omega \mid \omega \text{ in } \{(,)\}^* \text{ and } \omega \text{ is balanced}\}$

$= \{(), (()), ((()))), \ldots\}$

$(() \notin L$ Balanced parenthesis are those that can appear in arithmetic expressions

Example : A Regular language

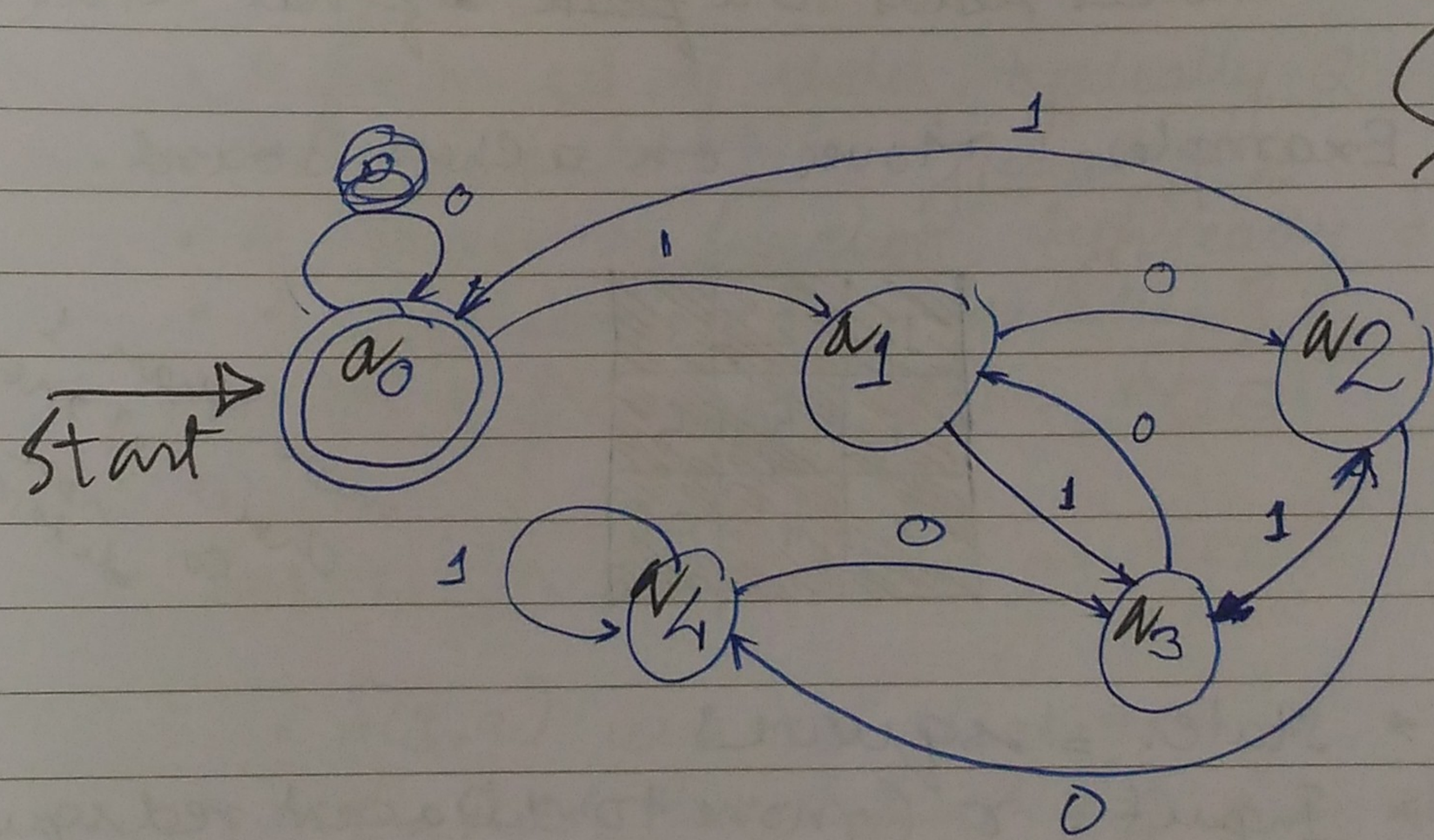
$L = \{\omega \mid \omega \text{ in } \{0,1\}^* \text{ and } \omega \text{ is viewed as a binary integer divisible by } 5\}$

$L = \{0, 101, 1010, 1111, \ldots\}$

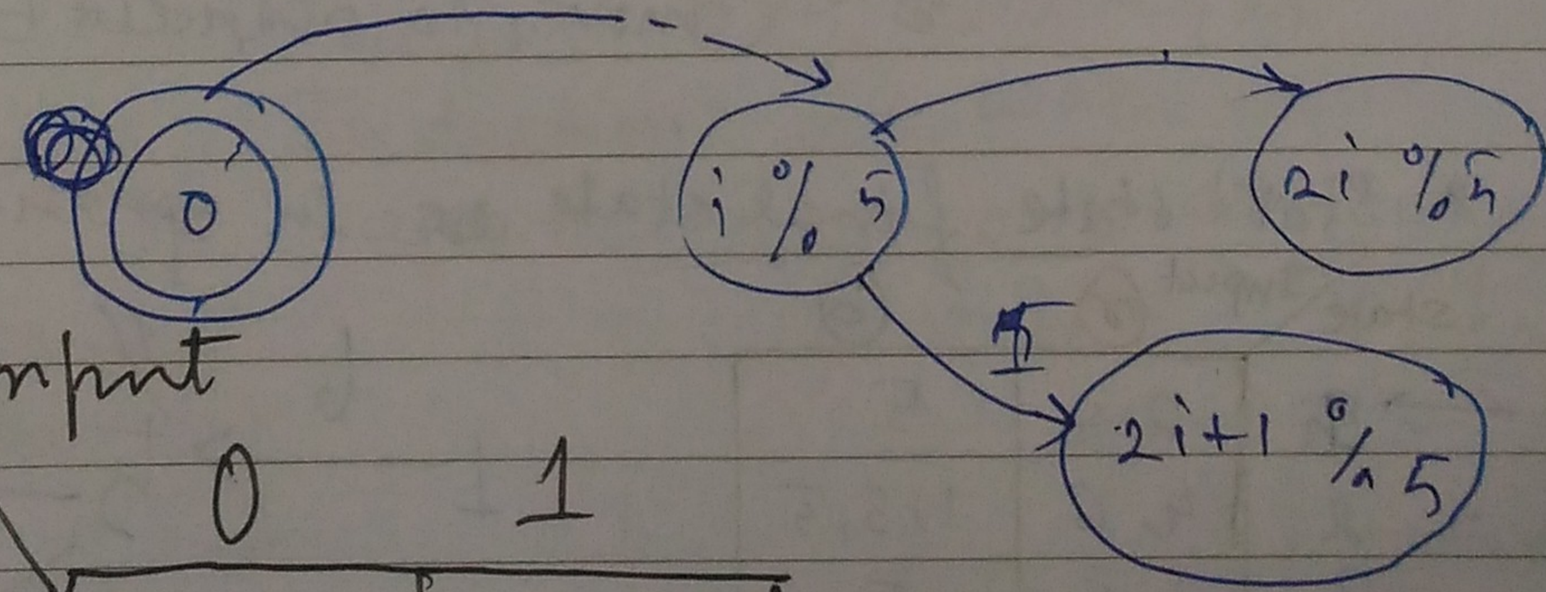
Corresponding to remainders by division by 5,
0, 1, 2, 3, 4.

DFA will atleast have 5 states.

- Start & only accept state is 0
- If string w represents an integer i, then ~~assum~~ assume $\delta(0, w) = i \% 5$
  - w0 represents 2i
    So we want $\delta(i\%5, 0) = 2i\%5$
  - w1 represents 2i+1
    So we want $\delta(i\%5, 1) = (2i+1)\%5$



State Transition Diagram //



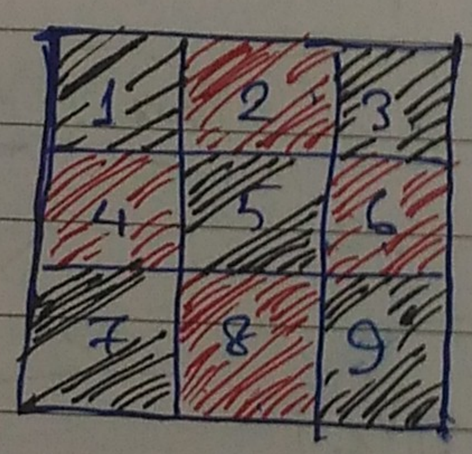
State \ Input	0	1
$\rightarrow^{*} q_0$	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_0
q_3	q_1	q_2
q_4	q_3	q_4

State Transition Table //

Date 27/7/17

Non-deterministic Finite Automata : (NFA)

- An NFA has the ability to be in several states at once
- Transitions from a state on an input symbol can be to any set of states
- An NFA starts in one start state
  - Accepts an input if any sequence of choices leads to a ~~finite~~ final state
- Example : Moves on a Chess Board.

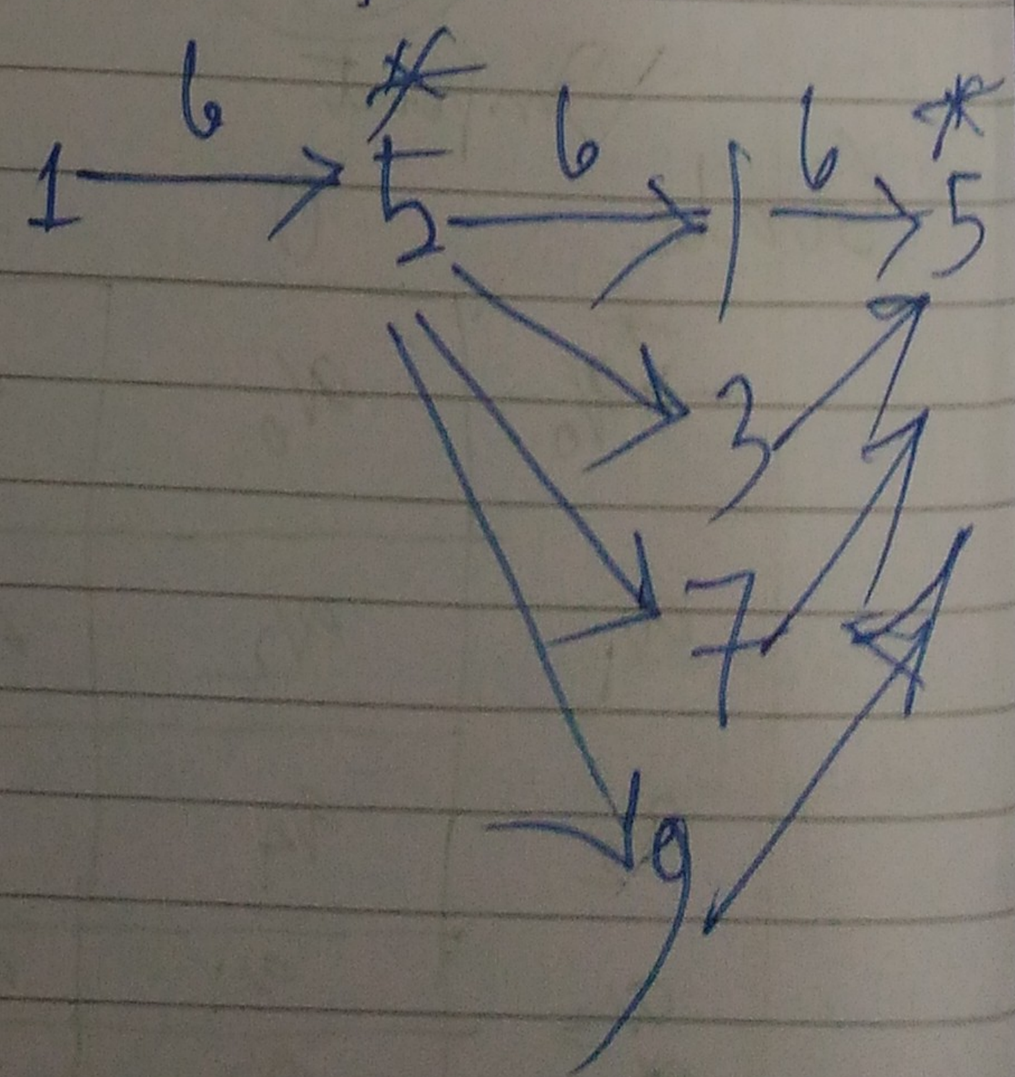


more than 1 choice of moving so non deterministic

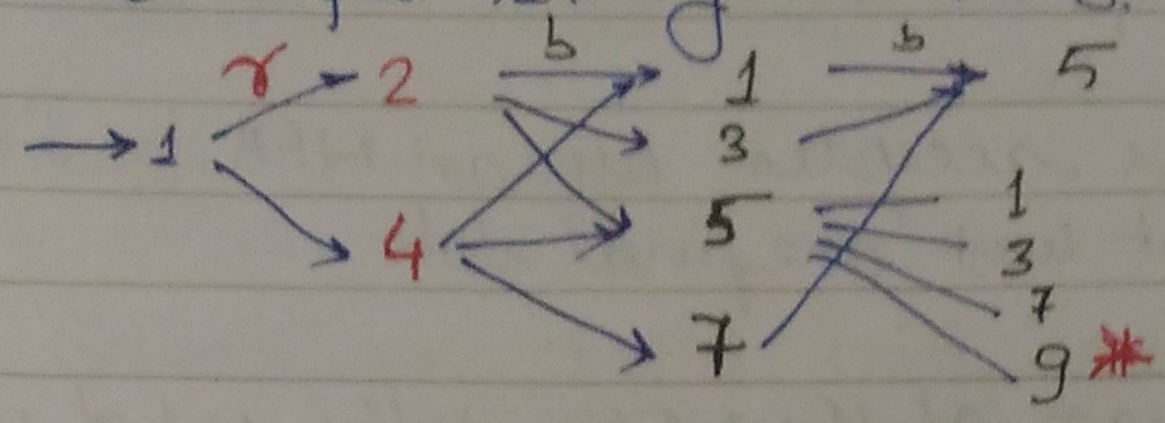
* States = squares
* Input = r (move to adjacent red square)
b (move to adjacent black square)

* Start state, final state are in opposite squares.

State \ Input	r	b
→ 1	2, 4	5
2	4, 6	1, 3, 5
3	2, 6	5
4	2, 8	1, 5, 7
5	2, 4, 6, 8	1, 3, 7, 9
6	2, 8	3, 5, 9
7	4, 8	5
8	4, 6	5, 7, 9
*9	6, 8	5



Let input string = r b b.



(1, 2, 5, 9*) (1, 4, 5, 9*)

r b b is accepted

atleast one path must lead to an accept state.

* Formal NFA

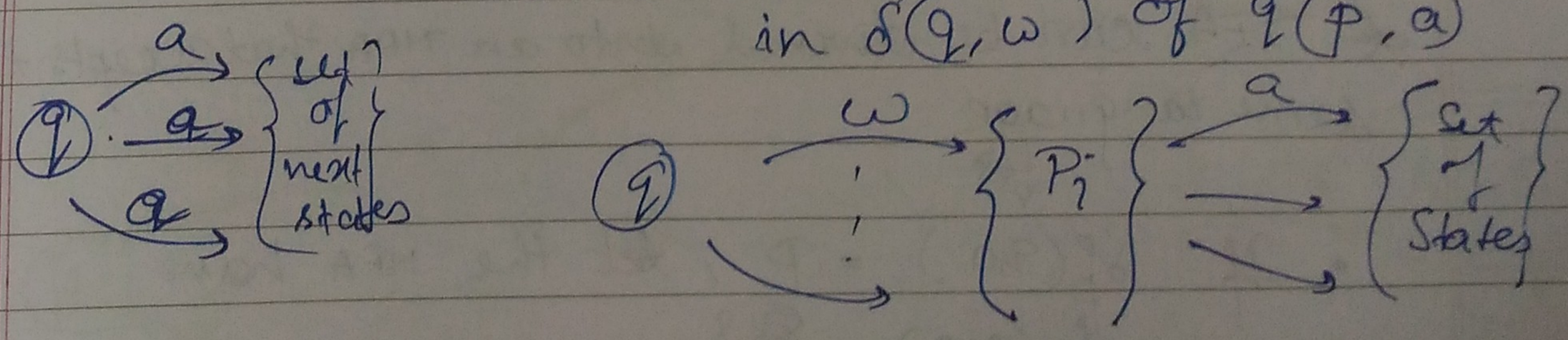
- A finite set of states, typically Q
- An input alphabet, typically Σ
- A transition function, typically δ
- A start state in Q, typically q_0
- A set of finite ~~sets~~ states $F \subseteq Q$

- Transition Function of an NFA
  - $\delta(q, a)$ is a set of states
  - Extend the definition of δ to strings.

    Basis: $\delta(q, \epsilon) = \{q\}$

    Induction: $\delta(q, wa)$

    = the union over all states p in $\delta(q, w)$ of $q(p, a)$



$$\delta(q, wa) = \delta(\underbrace{\delta(q, w)}_{\{P_i\}}, a)$$

$$= \bigcup_i \delta(P_i, a)$$

Date 2/8/17

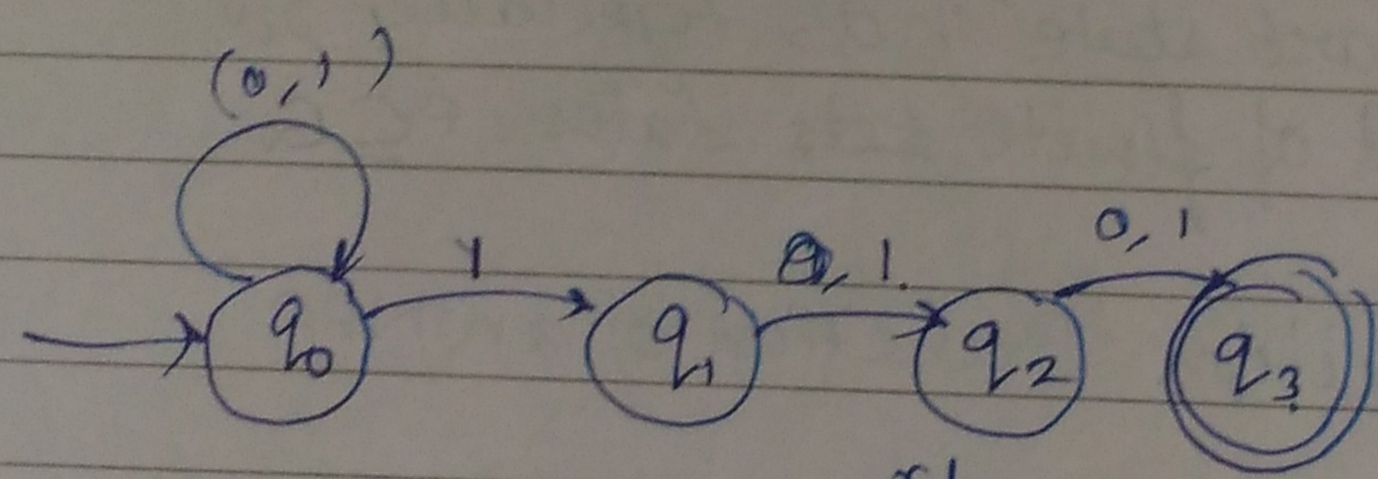
- Language of an NFA.

→ A string is accepted by an NFA if $\delta(q_0, a)$ contains at least one final state

→ The language of the NFA is the set of strings it accepts.

→ For our chessboard NFA, we saw that abb was accepted

→ If the input consists of only b's, the set of acceptible states alternates between $\{5\}$ and $\{1, 3, 7, 9\}$, so only odd ~~even~~ length non-empty strings of b's are accepted.



3rd Last digit of a number is 1.
100, 0100, 0111

Equivalence of DFA's, NFA's

A DFA can be turned into an NFA that accepts the same language

- If $\delta_D(q, a) = p$, let the NFA have $\delta_N(q, a) = \{p\}$

Then the NFA is always a set containing exactly one state - the state the DFA is in after reading the same input

- <u>Subset Construction</u> : A method for conversion of an NFA into an equivalent

~~DFA → NFA~~

Given an NFA with states Q, input Σ, transition function δ_N.
Start state q_0, and final state F,
construct equivalent DFA with:

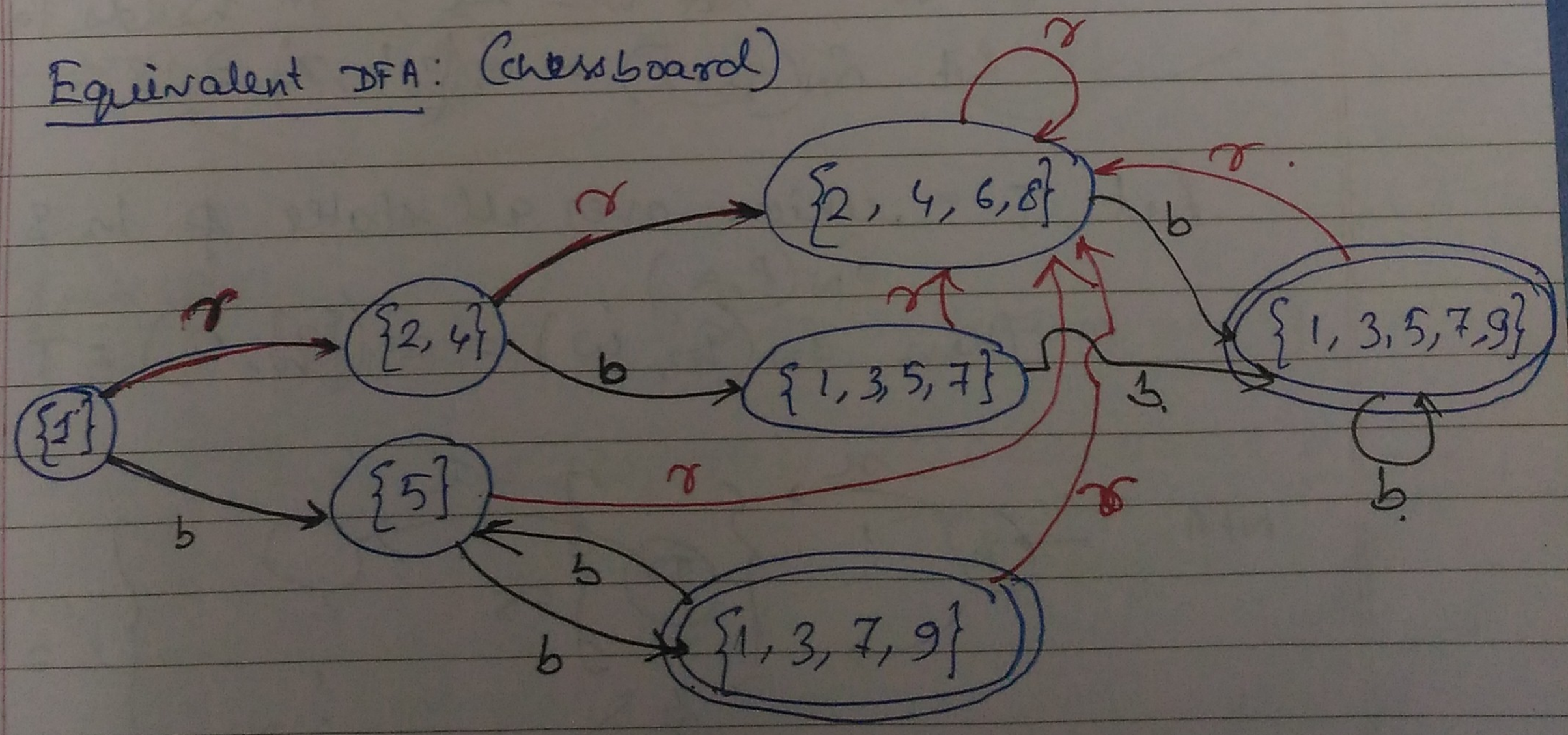
- States 2^Q (set of subsets of Q)
- Input Σ
- Start state $\{q_0\}$
- Final states = all those with a member of F

The DFA states have names that are states of NFA states eg: $\{p, q\}$ where p & q are two states of NFA

→ The transition function of δ_D is defined by
$\delta_D(\{q_1, \ldots, q_k\}, a)$
= the union over all $i = 1, \ldots k$ of $\delta_N(q_i, a)$

→ Example : Construct a DFA equivalent to "chessboard" NFA

<u>Equivalent DFA</u>: (chessboard)



State \ Input	a	b
→ {1}	{2, 4}	{5}
{2, 4}	{2, 4, 6, 8}	{1, 3, 5, 7}
{5}	{2, 4, 6, 8}	{1, 3, 5, 7, 9}
{2, 4, 6, 8}	{2, 4, 6, 8}	{1, 3, 5, 7, 9}
{1, 3, 5, 7}	{2, 4, 6, 8}	{1, 3, 5, 7, 9}
{1, 3, 7, 9}*	{2, 4, 6, 8}	{5}
{1, 3, 5, 7, 9}*	{2, 4, 6, 8}	{1, 3, 5, 7, 9}

Total there were $2^9 = 512$ states for DFA.
(9 states for NFA)
out of those 512 states we draw transition table in 9 states.

<u>Proof of Equivalence</u>: Subset construction.

→ Show by induction on $|w|$ that

$$\delta_N(q_0, w) = \delta_D(\{q_0\}, w)$$

Basis $w = \epsilon$

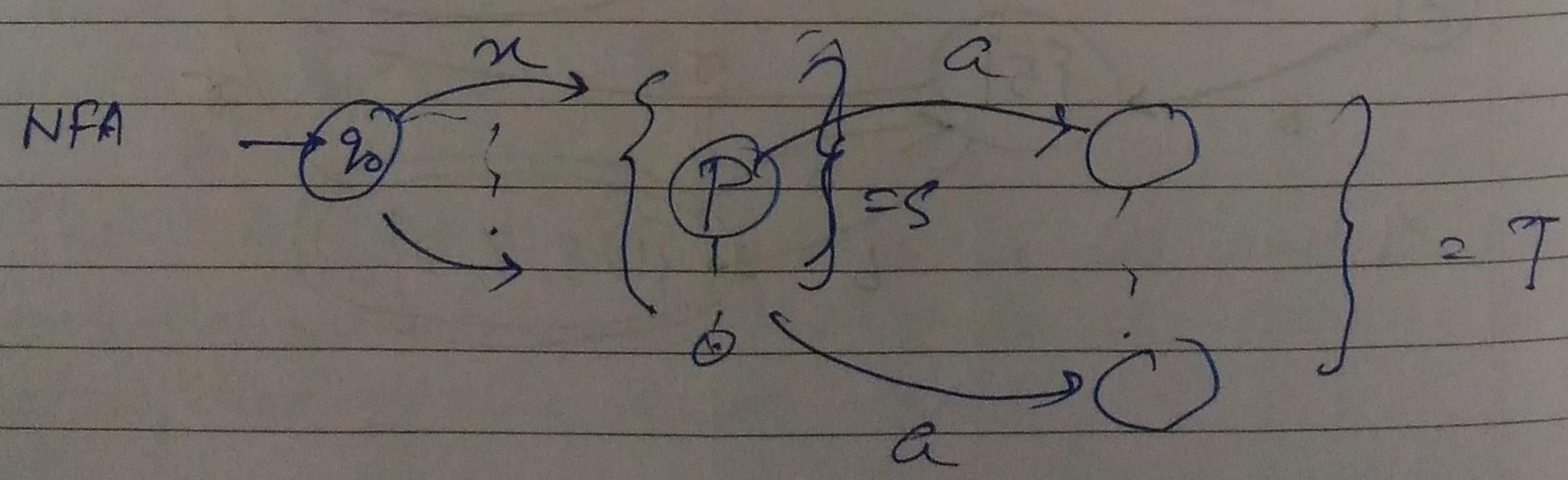
$$\delta_N(q_0, \epsilon) = \delta_D(\{q_0\}, \epsilon) = \{q_0\},$$

Induction: Assume IH for strings shorter than w, let $w = xa$; IH holds for x

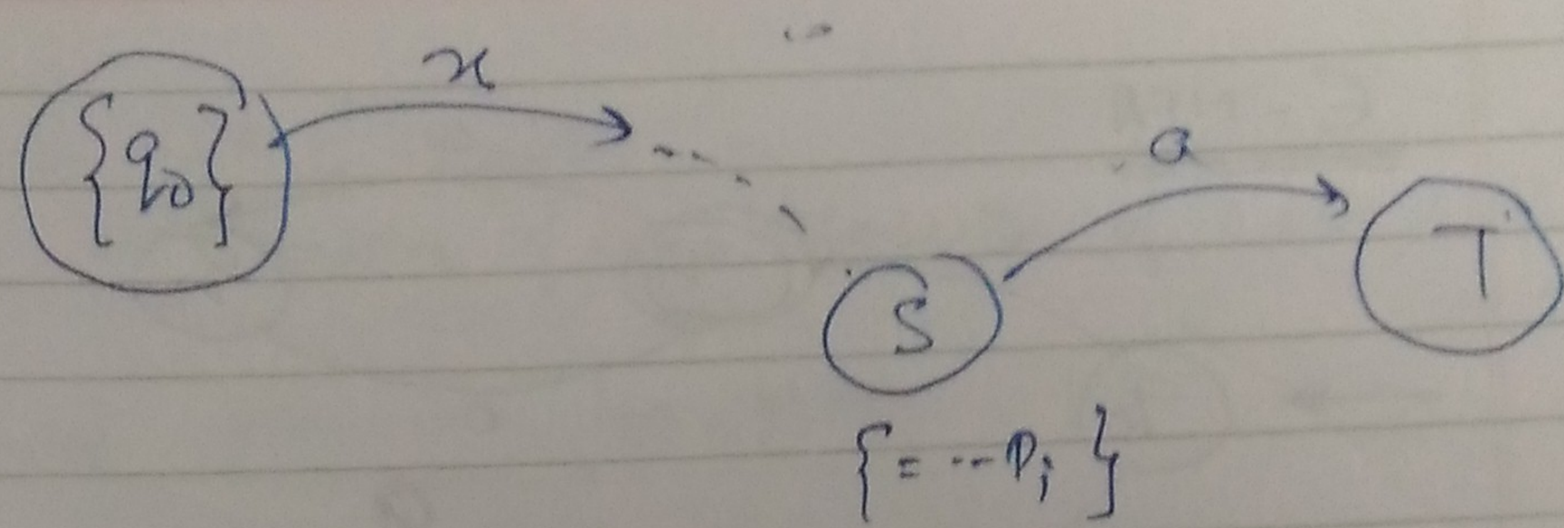
Let $\delta_N(q_0, x) = \delta_D(\{q_0\}, x) = S$

Let T = union over all states p in S of $\delta_N(p, a)$

Then $\delta_N(q_0, w) = \delta_D(\{q_0\}, w) = T$



DFA



NFAs with ∈ Transitions:

- We can allow state to state transition on ∈ input
- These transitions are done spontaneously without looking at the input.

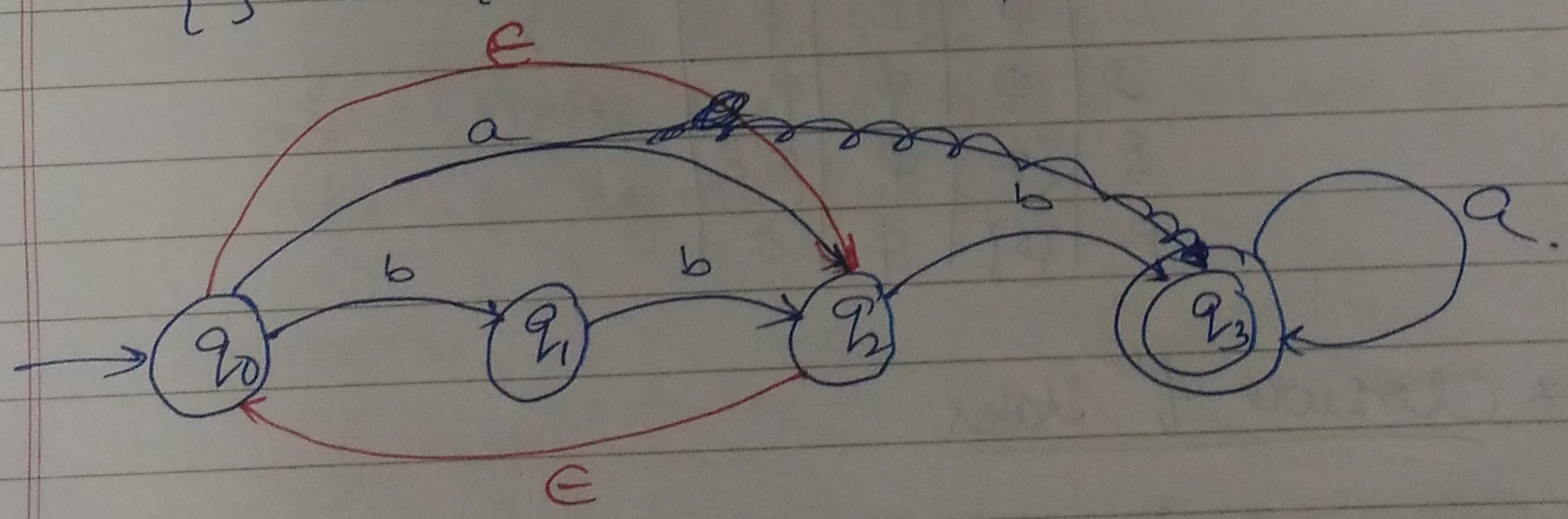
- Example

all possible combination.

$$L = \underbrace{\{0, bb\}^* \{b\} \{a\}^*}$$ Kleene closure

aba, abbba, abbaba, bbaa, b

$\{a\}^* \in \{\epsilon, a, aa, aaa\}$

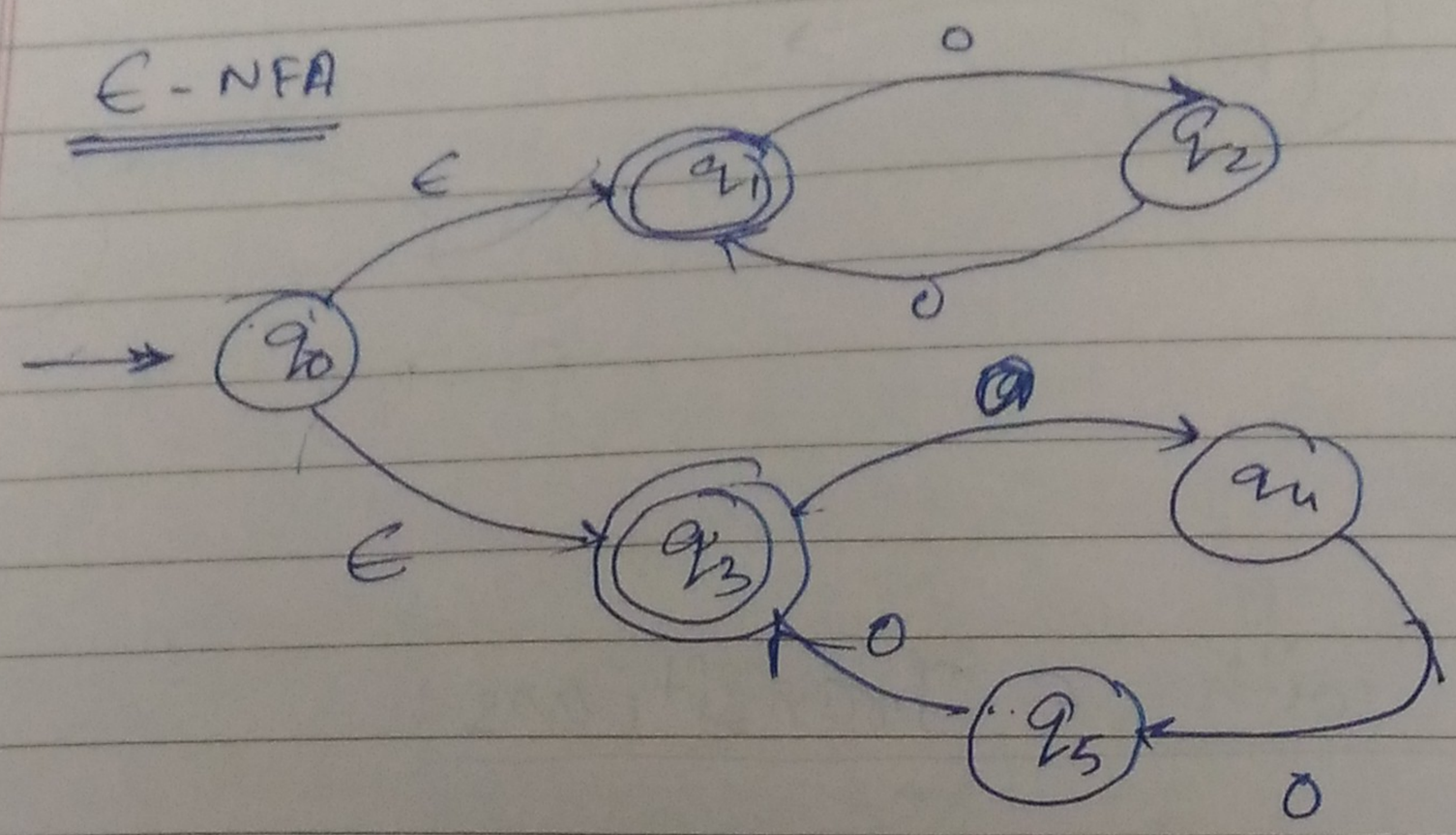


Example:

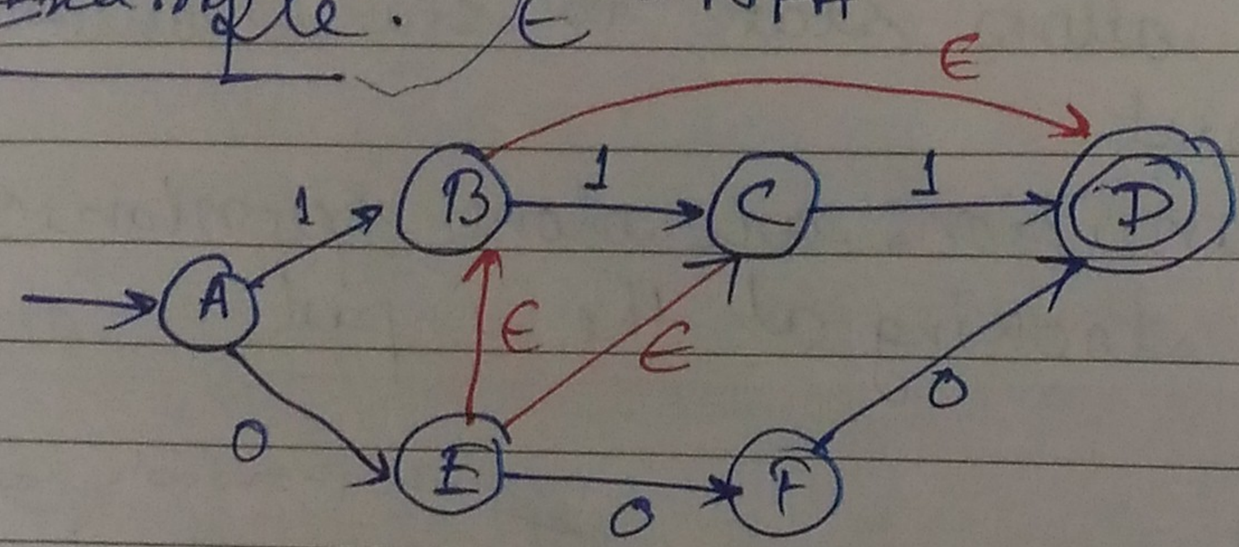
$L = \{0^k \mid k \text{ is a multiple of 2 or 3}\}$

$L = \{\epsilon, 00, 000, 0000, 00000, \ldots\}$

ϵ - NFA



Example: ϵ - NFA



State \ Input	0	1	ϵ
→ A	{E}	{B}	ϕ
B	ϕ	{C}	{D}
C	ϕ	{D}	ϕ
D	ϕ	ϕ	ϕ
E	{F}	ϕ	{B, C}
F	{D}	ϕ	ϕ

* Closure of States

- $CL(q)$ = set of states you can search from state q following only arcs labelled ϵ

- Example $CL(A) = \{A\}$
  $CL(E) = \{B, C, D, E\}$

ϵ is an empty string
ϕ is an empty set.

- Closure of set of states
  = union of the closure of each state.

* Extended delta for ϵ-NFAs

- Basis :
  $\hat{\delta}(q, \epsilon) = CL(q)$
- Induction :

  $\delta(q, xa)$ is computed as follows
  1. Start with $\delta(q, x) = S$ ← set of states
  2. Take the union of $CL(\delta(p, a))$ for all p in S

- Intuition :

  $\delta(q, w)$ is the set of states you can reach from q following a path labelled w.

  And note that $\delta(q, a)$ is the set of states for symbol a.

✓ Example :

Extended Delta :

$\delta(A, \epsilon) = \{A\}$

(E) → $\delta(A, 0) = \epsilon CL(E) = \{B, C, D, E\}$

$\delta(A, 01) = \delta(\delta(A, 0), 1)$

- Language of a ϵ-NFA is the set of strings w such that $\delta(q_0, w)$ contains a final state

$= CL(\delta(B, 1) \cup \delta(C, 1) \cup \delta(D, 1)) \cup \delta(E, 1))$

$= CL(\{C, D\} \cup \phi \cup \phi) = CL\{C, D\} = \{C, D\}$

* Equivalence of NFA, ε-NFA

* Every NFA is an ε-NFA
* It just have no transition on ε

converse requires us to take an ε-NFA and construct an NFA that accepts the same language

- We do so by combining ε-transitions with the next transition.

* Conversion from ε-NFA to NFA

- Start with an ε-NFA with states Q, input Σ, start state q_0, final states F, and transition function δ_E

- Construct an "ordinary" NFA with states Q, inputs Σ, start state q_0, final state F', and transition function δ_N.

- Compute $\delta_N(q, a)$ as follows:
  1. Let $S = CL(q)$
  2. $\delta_N(q, a)$ is the union over all p in S of $\delta_E(p, a)$

- F' = the set of states q such that $CL(q)$ contains a state of F

* Intuition :

δ_N incorporates ε-transitions before using the symbol a but not after.

Date 16/8/17

Summary

- DFAs, NFAs and ϵ-NFA's all accept exactly the same set of languages, the regular languages
- The NFA types are easier to design and may have exponentially fewer states than DFA
- But only a DFA can be implemented

Regular Expressions: (REs)

- are an algebraic way to describe languages
- describe exactly the regular languages
- If E is a regular expression then L(E) is the language it defines

* REs : Definition

- Basis 1: If a is any symbol then a is a RE, and $L(a) = \{a\}$
- Basis 2: ϵ is a RE and $L(\epsilon) = \{\epsilon\}$
- Basis 3: ϕ is a RE and $L(\phi) = \{\} = \phi$

Induction 1:

If E_1 and E_2 are regular expressions then $E_1 + E_2$ is a regular expressions and $L(E_1 + E_2) = L(E_1) \cup L(E_2)$

union of the language of E_1 & E_2

Induction 2:

Concatenation of language

If E_1 and E_2 are regular expression then E_1 , E_2 is a regular expression and $L(E_1 , E_2) = (E_1) . L(E_2)$

- Induction 3 :

If E is a RE then E^* is a RE and $L(E^*) = (L(E))^*$

* Precedence of operators.

* highest
.
\+ lowest

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

$L(0^* 1) = \{1, 01, 0001 \ldots\}$

$0 \quad L(0) = \{0\}$

$1 \quad L(1) = \{1\}$

$0+1 = \quad L(0+1) = L(0) \cup L(1)$

$\{0, 1\}$

Examples : REs.

$L(01) = \{0, 1\}$

$L(01+0) = \{01, 0\}$

$L(0(1+0)) = \{01, 00\}$

$L(0^*) = \{\epsilon, 0, 00, 000 \ldots\}$

$L((0+10)^*(\epsilon+1))$

= all strings of 0's & 1's without two consecutive 1's.

Equivalence of REs an Automata

- To show that for every RE there is an automation accepting the same language, pick the most powerful automation type : the ϵ-NFA

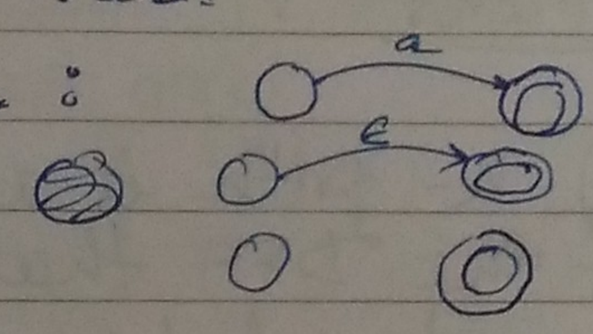
Date 23/8/17

- To show that for every automation there is a RE defining its language, pick the most restrictive type : the DFA

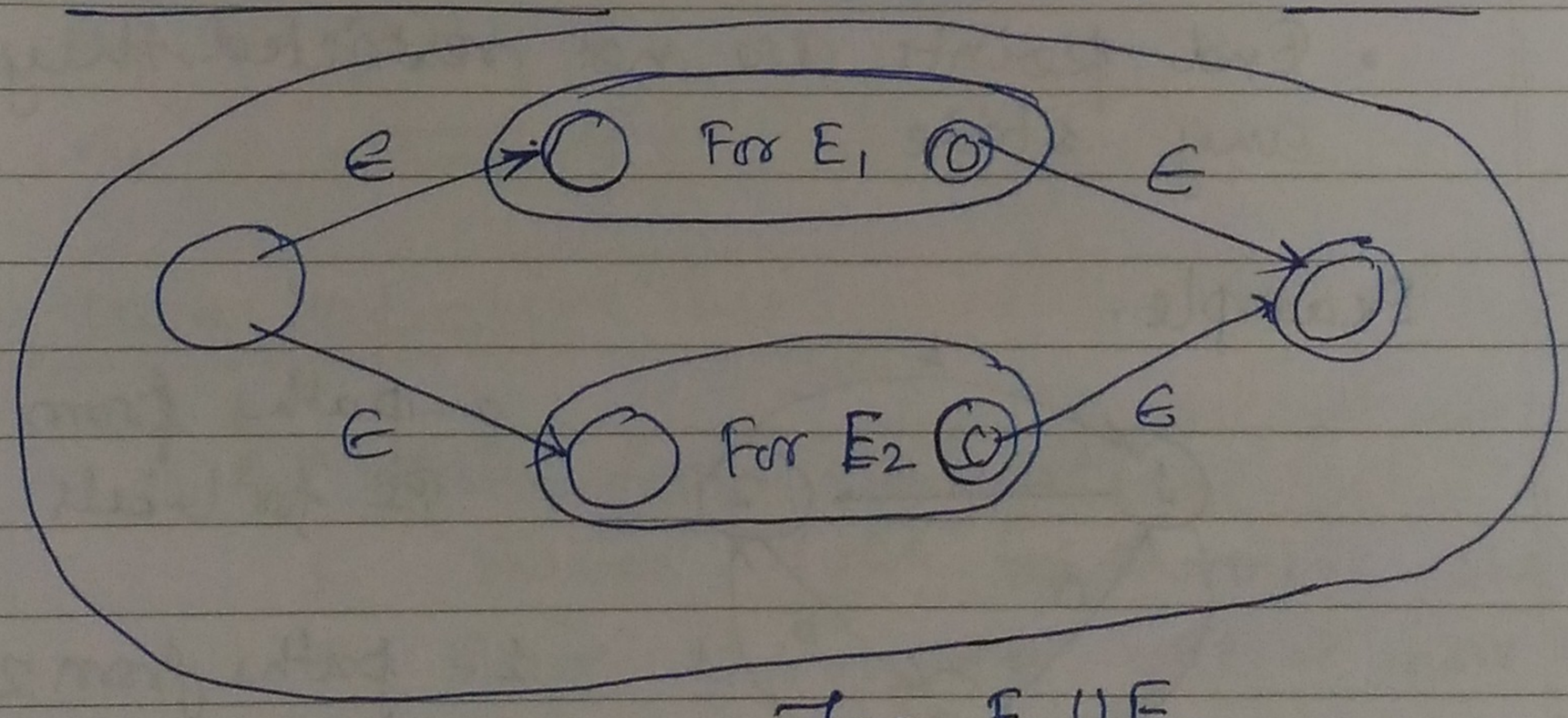
Converting a RE to an ε - NFA

- Proof is in induction on the number of operators (+, concatenation, *) in the RE

- RE to ε - NFA : Basis
  - Symbol **a** :
  - ε
  - φ

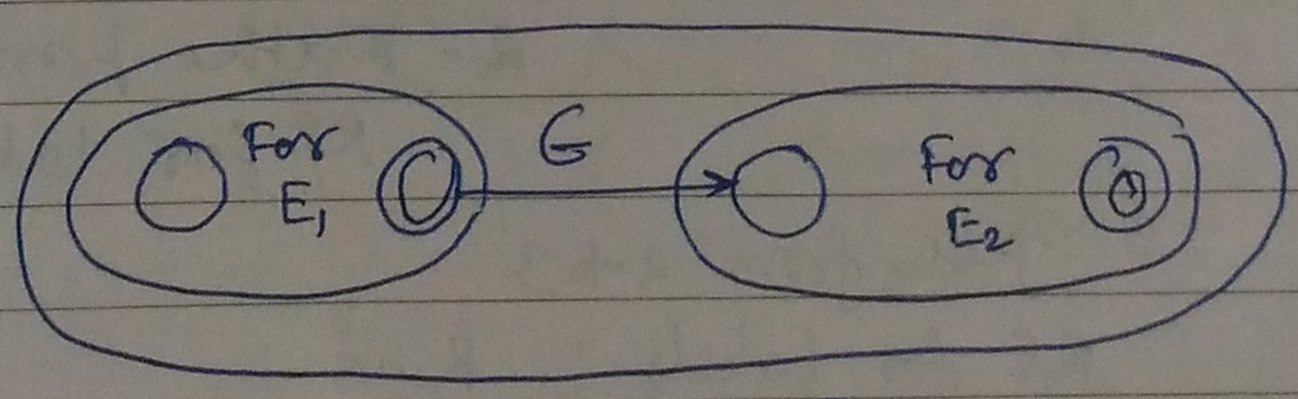


- RE to ε - NFA : Induction 1 - Union



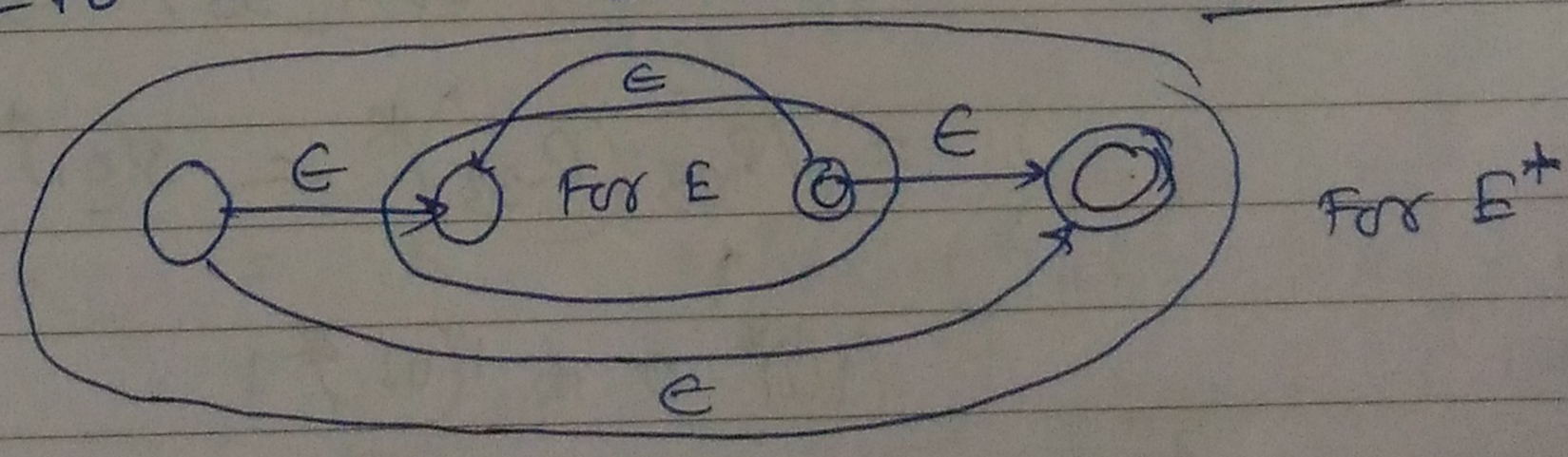
For $E_1 \cup E_2$

- RE to ε - NFA : Induction 2 - Concatenation



For $E_1 . E_2$

- RE to ε - NFA : Induction 3 - Closure



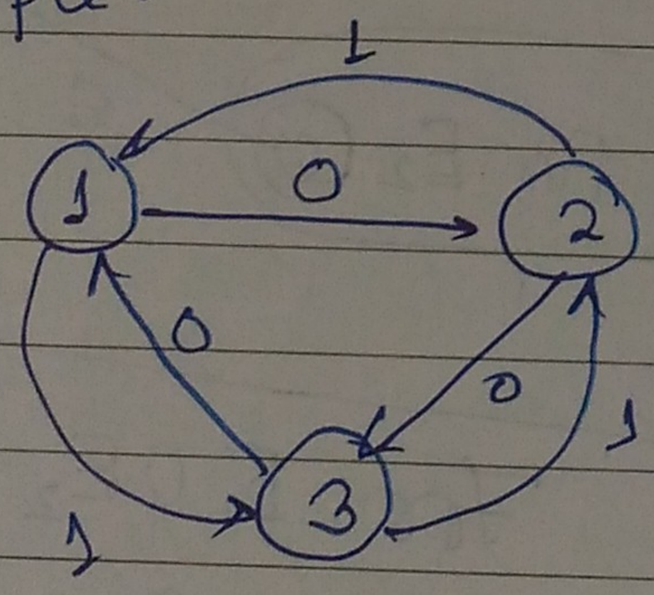
DFA to RE :

- States of the DFA are numbered as 1, 2, ... n
- We construct RE's for the labels of restricted sets of paths
  - Basis : Paths : single arcs or no arcs at all
  - Induction : paths that are allowed to traverse next state in order.

K- Paths

- A K- path is a path through the graph of the DFA that goes through no state numbered higher than K
- End points are not restricted, they can be any state.

- Example.



0 - paths from 2 to 3
RE for labels = 0

1 - paths from 2 to 3
RE for labels = 0 + 11

2 - Paths from 2 to 3
RE for labels : $(10)^*0 + 1(01)^*1$

3 . paths from 2 to 3
RE for labels : R_{23}^3

$$= R_{23}^2 + R_{23}^2 (R_{33}^2)^* R_{33}^2$$

$$= R_{23}^2 (R_{33}^2)^* \leftarrow 0(01)^*(1+00) + 1(10)^*(0+11)$$

$$\uparrow$$

$$(10)^* 0 + 1(01)^* 1$$

* K-Path Induction

- Let R^k_{ij} be the regular expression for the set of labels of k-paths from state i to state j

- **Basis:**

$K = 0$

R^0_{ij} = sum of labels of arcs from i to j
  - ϕ is no such arc
  - But add ϵ if $i = j$

- Example : Basis

$R^0_{12} = 0$

$R^0_{11} = \cancel{\epsilon + \phi}\ \phi + \epsilon = \epsilon$

- **K-path Inductive Case:**

  * A K-path from i to j either
    1. Never goes ~~to~~ through state k
    2. Goes through k one or more times.

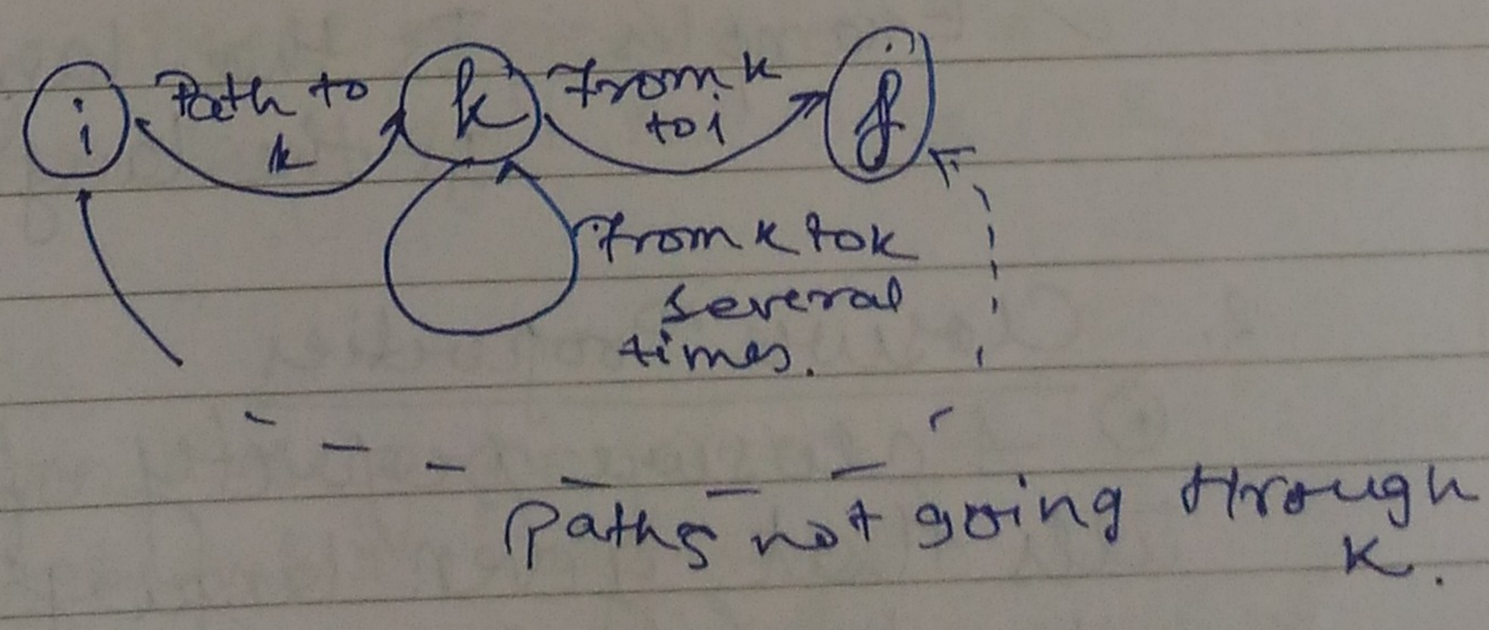


Illustration:

$$R^k_{ij} = R^{k-1}_{ij} + R^{k-1}_{ik} \left(R^{k-1}_{kk}\right)^* R^{k-1}_{kj}$$

Not through k

i to k 1st time.

0 or more times.

k to j

* Final Step:

The RE with the same language as the ~~lang~~ is the sum (union) of R_{if}^{n}, where

1. n is the number of states; i.e, paths are unconstrained.
2. i is the start state
3. f is one of the final states.

* Properties of Language Classes.

- A language class is a set of languages eg: the regular languages, context free languages, context sensitive languages, etc.
- Language classes have two important kinds of properties
  1. Decision properties
     - ✓ An algorithm that takes a formal description of a language (eg: a DFA) and tells whether or not some property holds
     - ✓ Examples: Is the language L empty
       Is the language L infinite
  2. Closure Properties
     - ⊙ A closure property of a language tells that given languages in the class an operator (eg, union) produces another language in the same class.
     - ⊙ Example: The regular languages are obviously closed under union, concatenation and (Kleen) closure.

31/8/17

~~What is Regular Languages~~

The Emptiness problem:

- Given a regular language, does the language contain any string at all.

- Assume the representation is DFA
- Construct the transition graph
- Compute the set of states reachable from the start state
- If any final state is reachable then yes, else no.

The Infiniteness Problem:

- Is a given regular language infinite?
  1. Eliminate states ~~that~~ ~~are~~ reachable from the start state
  2. Eliminate states that do not reach a final state
  3. Test if the remaining transition graph has any cycles.

* Pumping Lemma for regular languages

For every regular Language L — no. of states of DFA for L
There is an integer n such that for every ~~integer~~ string ω in L of length $\geq n$,
we can write $\omega = xyz$ such that — labels along first cycle on path labelled ω
1. $|xy| \leq n$
2. $|y| > 0$
3. For all $i \geq 0$, xy^iz is in L

Assume regular languages are closed under $\cap$

$L_2 \cap L(0^* 1^*) = L_1 = 0^n 1^n$

If L_2 were regular then $L_2 \cap L(0^* 1^*)$ would be but isn't

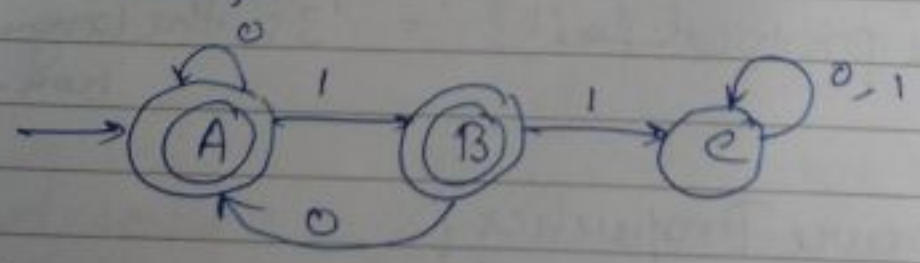
(not possible to check no. of 0's = no. of 1's)

- <u>The Membership Problem:</u>

  - Our first decision property is the question "Is string w in regular language L?"

  - Assume L is represented by a DFA A
    - Simulate the action of A on the sequence of input symbols forming w.

<u>Example:</u>

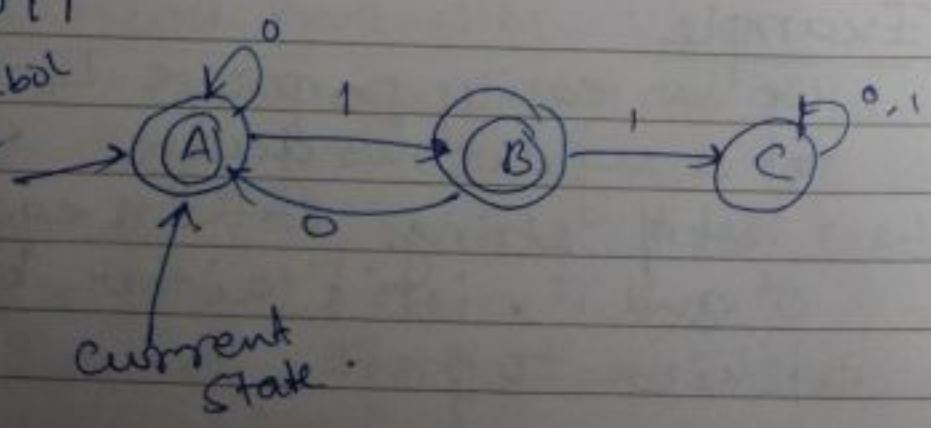


- Accepts all strings without two consecutive 1's.

$A \to B \to A \to B \to C$.

$w = 01011$

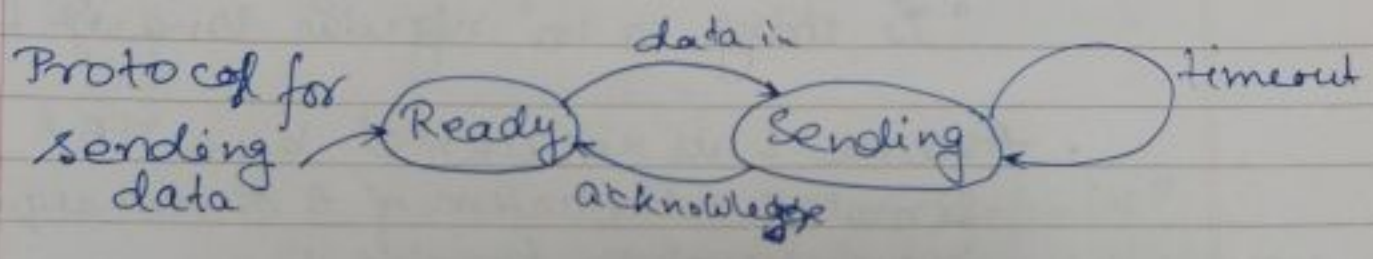
next symbol



Use the RE representation of languages

- <u>Why Decision Properties?</u>
  - When we talk about protocols represented as DFAs, we find that important properties of a good protocol are related to the language of the DFA.

- <u>Example</u>:



"Does the protocol terminate?" = "Is the language finite?"

"Can the protocol fail?" = "Is the language non-empty"

- <u>Why Closure Properties?</u>

1. Helps construct representations.
2. Helps show languages not to be in the ~~class~~ class

<u>Example</u>:

We can easily ~~pa~~ prove $L_1 = \{0^n 1^n \mid n \geq 0\}$ is not a regular language.

L_2 = set of strings with an equal number of 0^s and 1^s. isn't either but that fact is trickier to prove

- Example : Use of pumping lemma
  - we have claimed $\{0^k 1^k \mid k \geq 1\}$ is not a regular language
  - Suppose it were,
    Then there would be an associated n for the pumping lemma
  - Let $w = 0^n 1^n$
    we can write $w = xyz$ where x & y consist of 0's and $y \neq \epsilon$ $|xy| \leq n$
  - But then xy^2z would be in L and this string has more 0's than 1's

* Decision Property : Equivalence

- Given regular languages L and M, is L = M?
- Algorithm involves constructing the product DFA from DFA from DFA's for L & M
  - Let these DFA's have sets of states Q & R respectively
  - Product DFA has set of states $Q \times R$, i.e pairs $[q, r]$ with q in Q, r in R
  - Start state $[q_0, r_0]$ (the start states of DFA's for L, M)
  - Transitions :
    $\delta([q, r], a) = [\delta_L(q, a), \delta_M(r, a)]$ where δ_L & δ_M are the transition functions for the DFA's of L, M.

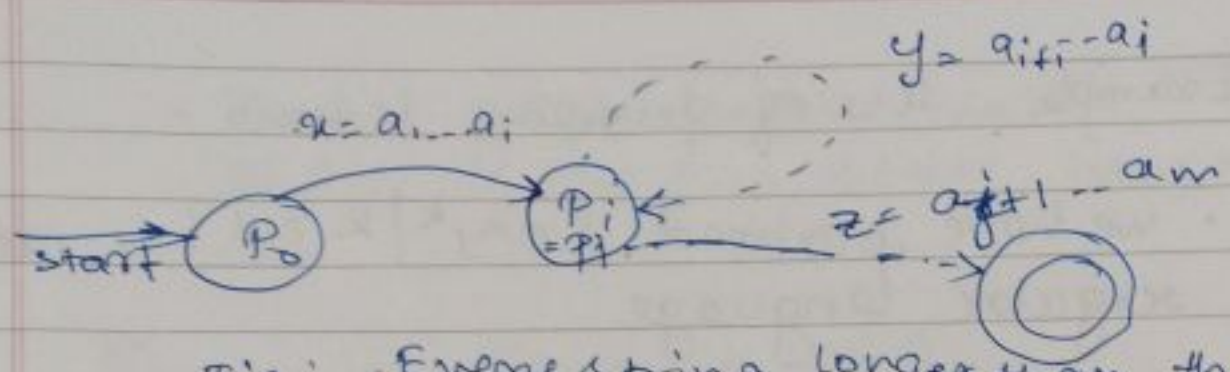


Fig: Every string longer than the no. of states must cause a state to repeat

Note that x may be empty in the case that $i = 0$

Also z may be empty if $j = n = m$.

However, y cannot be empty, since i is strictly less than j

Now consider what happens if the automaton A recieves input xy^*z for $k \geq 0$

<u>For $k = 0$</u>

The automaton goes from the start state q_0 (which is also P_0) to P_i on input x. Since P_i is also P_j; it must be that A goes from P_i to the accepting state (shown in Fig)

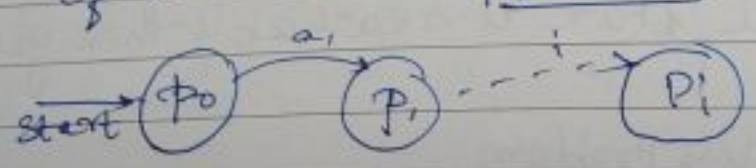
<u>For $k > 0$</u>

A goes from q_0 to P_i on input x circles from P_i to P_i <u>k times</u> on input y^k, and then goes to the accepting state on input z. Thus for $k \geq 0$, xy^kz is also accepted by A_j, that is xy^kz is in L

Proof: Let the DFA A has n states

W is in L(A) and $w = a_1 a_2 \ldots a_m$ where $m \geq n$ q_0 = start state of A

For $i = 0, 1, \ldots n$ define state p_i to be $\delta(q_0, a_1, a_2, \ldots a_i)$, where δ is the transition function of A. That is, p_i is the state A is in after reading the first i symbols of w. Note that $\boxed{q_0 = p_0}$



By the pigeon hole principle, it is not possible for (n+1) different p_i's for $i = 0, 1, \ldots n$ to be distinct, since ~~the~~ there are only n different states.

Thus we can find two different ~~differ~~ integers i and j, with $0 \leq i < j \leq n$ such that $p_i = p_j$

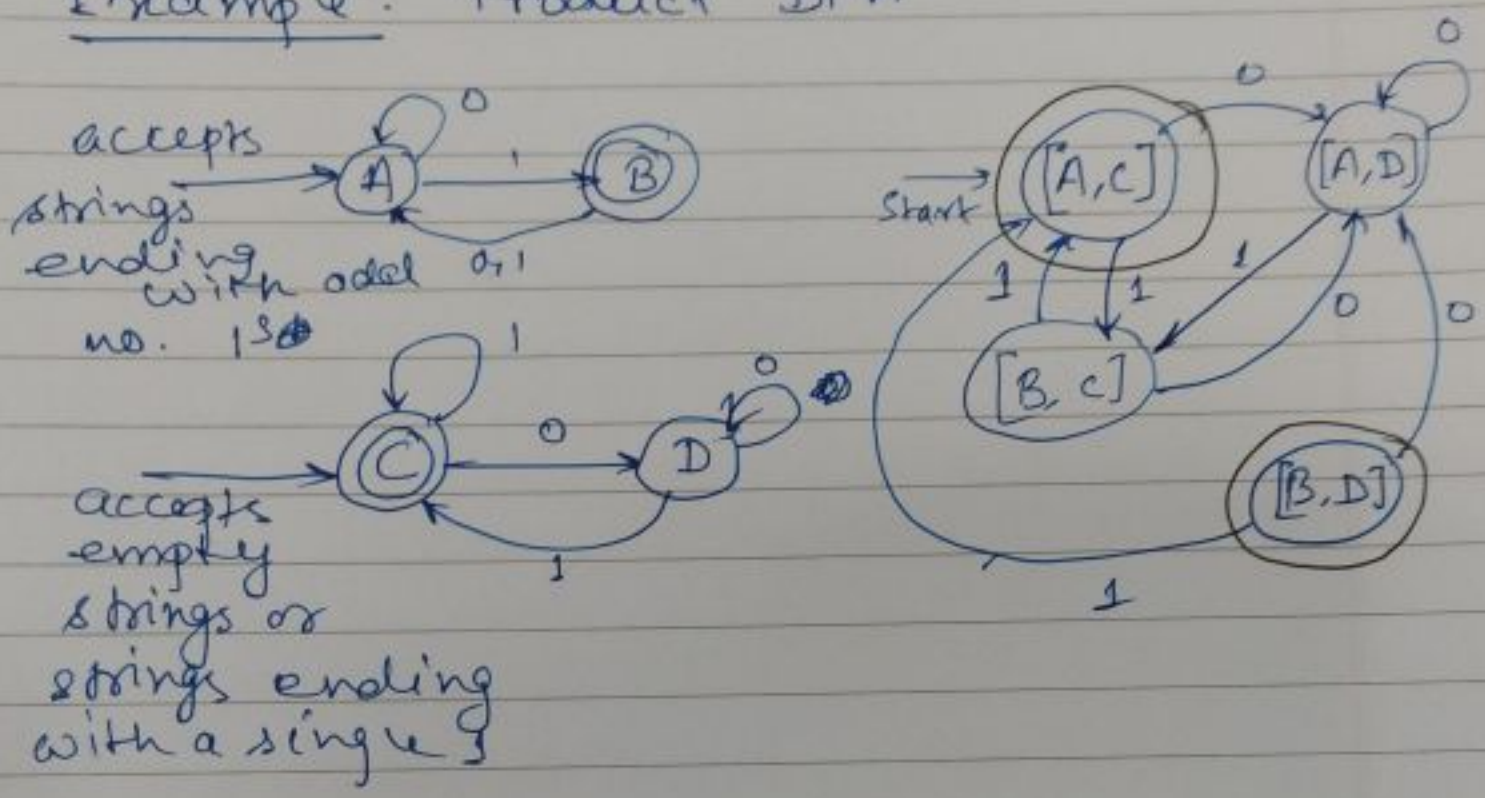
Now we can break $w = xyz$ as follows

1. $x = a_1 a_2 \ldots a_i$
2. $y = a_{i+1} a_{i+2} \ldots a_j$
3. $z = a_{j+1} a_{j+2} \ldots a_m$

That is x takes us to p_i;
y takes us from p_i back to p_i (since p_i is also p_j)
and z is the balance of w

- That is we ~~simulated~~ simulate the two DFA's in the two state component of the product DFA.

<u>Example</u>: Product DFA



- Make the final states of the product DFA be those states $[q, r]$ such ~~that~~ that exactly one of q & r is a final state of its own DFA.

- Thus the product accepts w iff w is in exactly one of L & M.

The product DFA's language is empty iff $L = M$.