FORMAL LANGUAGES & AUTOMATA THEORY (FLAT)

19/7/18

First part of compilation process → takes a string of symbols as input

↓

outputs ACCEPT/REJECT (on whether the string was syntactically correct or not)

In case of sequential ckt., response of the machine not only depends on the current i/p, but also the current state of the machine.

→ receives a binary string of i/p symbols
→ has a finite no. of states
→ responds based on the i/p it has received & current state of the machine.

A practical/real-world machine can't hv an infinite no. of states.

Automata → Greek word; meaning self-acting machine (singular. automaton)

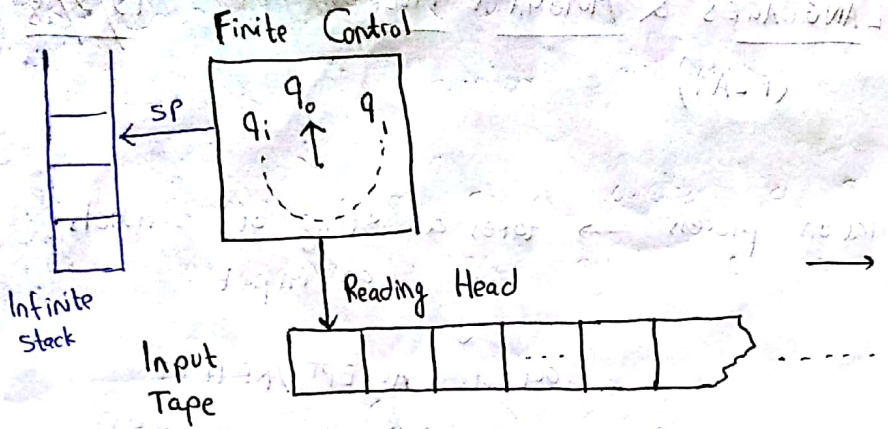
All these machines are mathematical machines; not conventional machines that convert one form of energy to another.

Mathematical Machines — Models of computation
↳ whatever is done through an algorithm is a computation.

↓

Major Features:

→ accept i/p as strings of symbols
→ have finite no. of internal states.
→ don't have output units (screen, etc.) as such.
⇒ just answers ACCEPT/REJECT
→ no such i/p devices ( ) are available.
Only infinitely long i/p tape is available. From this, MM read, symbol by symbol from i/p tape through its reading head



Capability of machine can be increased by adding a stack pointer & an infinite stack.

Ultimate objective of automata theory is to classify the problems which can be solved through algo & ones which can never be solved through algorithmic procedures.

Question is if there is any problem which can't be solved through algorithms. / is this possible to solve all probs. by forming algorithms.

First we need to check if prob. is solvable (not unsolved)

Eg. Can there be an algo (ALGO) s.t.

(Algo$_i$, i) → ALGO → whether Algo$_i$ will eventually halt on input i

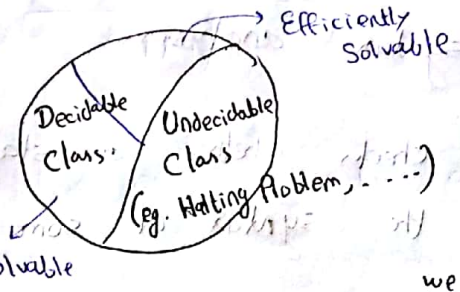
↓ description of another algo, Algo$_i$ with same input i

HALTING PROBLEM

It can be proved, no algo procedure will never exist for Halting Problem. (i.e. never be solved with an algorithmic procedure)

Such type of probs. are called UNDECIDABLE PROBLEMS

So, we want to divide class of All Problems into



If we want to test decidablity/undecidability, we must hv some machine (abstract) where we must be capable of supplying as much amt of memory & allow the m/c to run for ~~any~~ infinite amt of time.

Thus no practical machine can't.

Another ultimate objective : to find out whether the problem is efficiently solvable, or just solvable (for decidable class)

These probs. can't be judged w/o abstract machines.

All abstract machines are not equally capable. Simplest of such, is ① FINITE AUTOMATA (FA)

↓

Simply chcks whether a supplied string follows a specific pattern

Eg: For checking whether a variable is properly named, keywords (int, floats ...) are properly spelt. for verifying this by a compiler prog, a machine is used ~~called~~, LEXICAL ANALYZER (LA)

eg.

↓

(simplest)

→ no stack like memory

→ finite no. of states

→ infintely long i/p tape from which it can read symbols one by one.

LA is ~~simplest~~ an eg. of the simplest type of abstract machine, FA.

① FINITE AUTOMATA (FA)

eg. Lexical Analyzer

② Push Down Automata (PDA)

→ used for syntax analysis

↓

compiler prog. checks whether a statement is written according to the syntax of some language.

Eg: if (a == b) c = d ;

✗ if (a = b) c = d ;

Thus, using these abstract machines, some practical application

Eg: Syntax Analyzer ...

③ Turing Machine

most powerful.

Not only ~~write on~~ read from the i/p tape, but also can write on the tape.

Church's Hypothesis / Church - Turing Thesis :

Main pillar on which the entire subj rests.

Any "effective computation" whatsoever, any algorithmic procedure that can be carried out by a human being or a team of human beings or a computer can be carried out by some Turing Machine.

⇒ So for some prob., if we can show no Turing m/c exists, we can conclude that no algo exists for that problem

⇒ UNDECIDABLE CLASS

Formal Lang.

Formal → very precise defⁿ ; free from ambiguity of natural lang.

Lang. → defined by a set of grammar. (rules)
sentences may be finite / inf.

Every abstract m/c basically accepts a countably infinite set of strings, i.e. language (in Mathematics)

if there's a lang. ⇒ there's grammar (formal rules)

All the abs. m/c can have alternative models, Formal Lang based models / Grammar Based Models.

So these two, Formal Lang and Automata are closely related.

25/7/18.

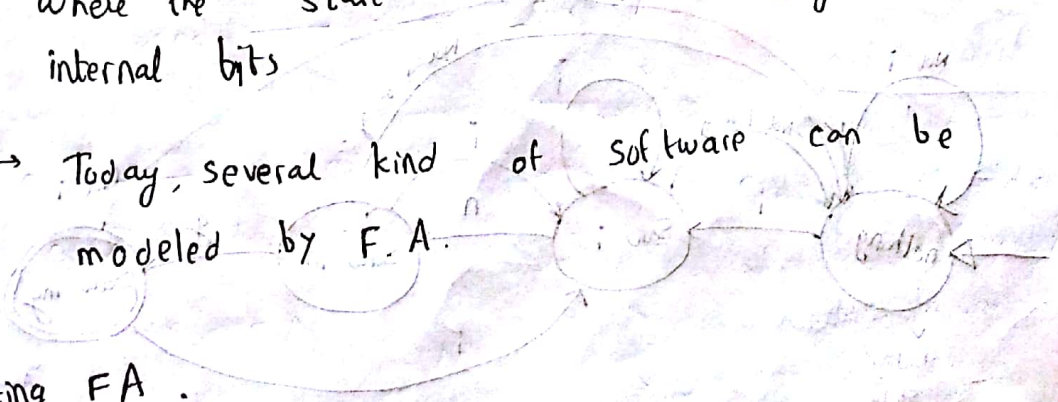
Finite Automata :

- An Informal Defⁿ.

→ are finite collection of states with transition rules that take you from one state to another.

→ original application was sequential switching circuits, where the "state" was the setting of internal bits

→ Today, several kind of software can be modeled by F.A.

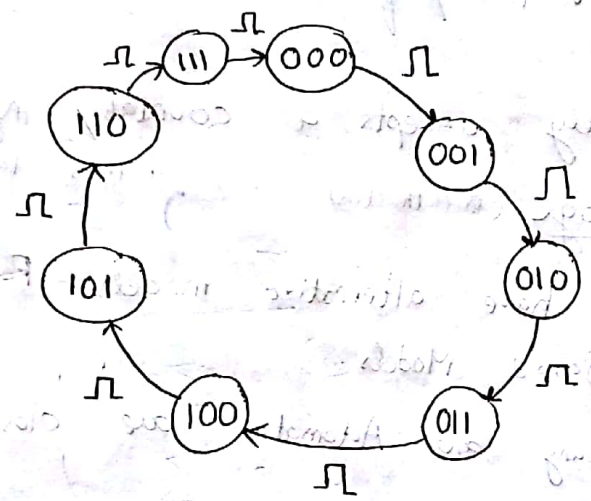


Representing FA :

→ Simplest representation is often a graph.

- Nodes = States
- Directed Arcs indicate state transition
- Labels on arcs tell what causes the transition

3-bit BINARY UP COUNTER



The State diagram for a 3-bit Binary UP Counter

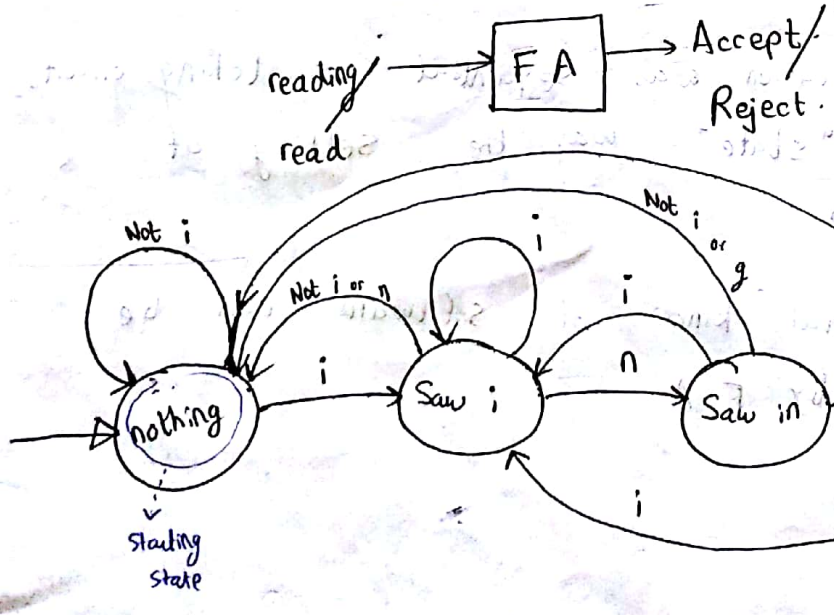
→ finite no. of states with transition rules that moves automata from one state to another state, as per the i/p.

→ Practical eg. of a finite automata

→ States are basically memory elements of the abstract machine

→ Next state depends on the current state and the input.

Eg: Recognizing Strings Ending in "ing"



Automata has to memorize it has received a symbol 'i'
⇒ memorize it has received 'i' and subsequent symbol 'n'
⇒ So informally, it must have at least 3 states

In a finite automata, at least one is spcly. marked as ACCEPT state / FINAL state

After reading an entire i/p string, if it moves to an accept state, then string is accepted by m/c

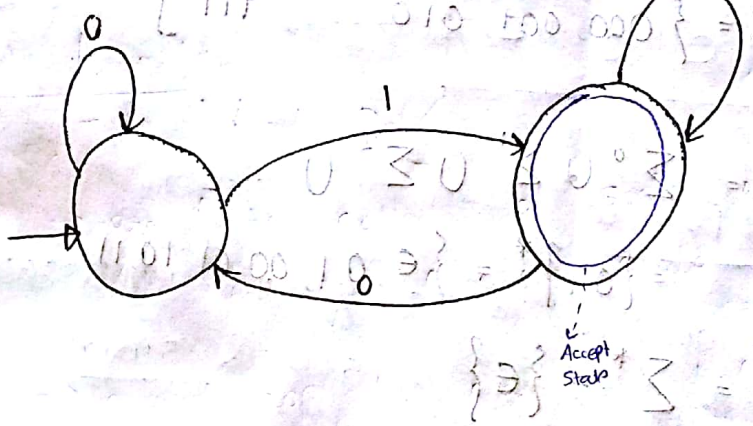
... if some non-accept state, ... rejected by m/c

When m/c starts reading the i/p string, it is assumed to be in Starting State

Eg: FA accepting all Binary Strings ending with a 1 or more 1's

{1, 01, 011, 111, ...}

~~0~~0, 0~~10~~0,



Eg: FA accepting all binary strings in which both the no. of 0's and 1's ~~are~~ even

HW.

ALPHABET

→ is any finite set of symbols

→ Eg: ASCII, Unicode, $\{0,1\}$, (binary alphabet), $\{a,b,c\}$, etc.

STRINGS:

→ A string is a finite sequence of symbols chosen from some alphabet

→ ϵ stands for empty string i.e. $|\epsilon| = 0$

→ Eg: Let $\Sigma = \{0,1\}$

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

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

$$\Sigma^2 = \{00, 01, 10, 11\}$$

$$\Sigma^3 = \{000, 001, 010, \ldots, 111\}$$

$$\vdots$$

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$$

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

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

LANGUAGES

→ A language is a set of strings all of which are chosen from some Σ^*, where Σ is a particular alphabet.

→ In automata theory, a problem is the question of deciding whether a given string (ω in Σ^*) is a member of some particular language (ω is in L?)

DETERMINISTIC FINITE AUTOMATA (DFA)

26/7/18

Formal defn for DFA

→ A formalism for defining languages, consisting of:

① A finite set of states (Q, typically)

② An input alphabet (Σ, typically) (finite set of symbols, each of which can form strings)

③ A transition function (δ, typically)

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

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

④ A start state (q_0 in Q typically) ← (q_0 must be an element of Q.)

⑤ A set of final states ($F \subseteq Q$, typically)

⊛ Final and Accepting states are synonyms.

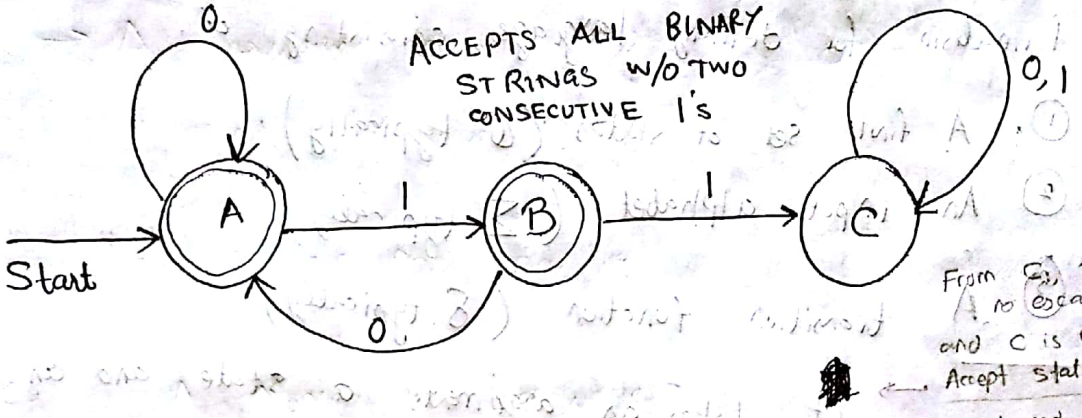
→ In proofs, we often talk about a DFA in "five tuple" notation:

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

→ Graph Representation of DFA's:

- Nodes = states
- Arcs represent transition functions
- Arrows labelled "Start" to start state
- Final states indicated by double circles

Eg: Graph of a DFA



From C, there is no escape and C is not a Accept state.

If at end, m/c is in state A or B, then no consecutive 1's

⇒ Alternative Representation

Transition Table

State	Input Symbol 0	1
→ *A	A	B
*B	A	C
C	C	C

Finite states labelled with asterisk (*)

Extended Transition Function

→ We describe the effect of a string of inputs on a DFA by extending δ to a state & string.

→ Induction on length of string

→ Basis: for 0 length string

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

→ Induction:

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

* w = a string
  a = an i/p symbol

Eg: Extended Delta (from prev graph)

$$\delta(B, 011) = \delta(\delta(B, 01), 1)$$

(i.e after some time it is in state B)

$$= \delta(\delta(\delta(B, 0), 1), 1)$$

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

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

∴ C is the state, which the m/c will finally move after reading the i/p signal 011.

Language of a DFA :

→ Automata of all kinds define languages

→ If A is an automaton, L(A) is its language

→ For a DFA A

L(A) = the set of strings labeling paths from the start state to a final state

→ Formally :

L(A) = the set of strings w such that $\delta(q_0, w)$ is in F.

Eg:

The language of our example DFA is :

$\{ w \mid w \text{ is in } \{0,1\}^* \text{ and } w \text{ does not have two consecutive 1's} \}$

Regular Languages :

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

Some languages are not regular.

Eg: A Nonregular Language

$$L_1 = \{ 0^n 1^n \mid n \geq 1 \} = \{01, 0011, 000111, \ldots\}$$

Another Eg: :

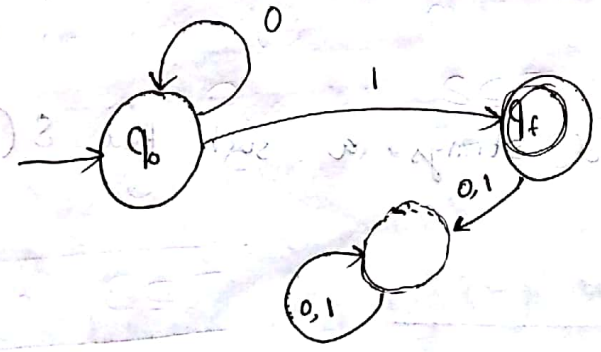
$\Sigma = \{(,)\}$

(Language of balanced parenthesis) $L_2 = \{ w \mid w \text{ in } \{(,)\}^* \text{ and } w \text{ is balanced} \}$

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

$L = \{ 0^n 1 \mid n \geq 1 \} = \{01, 001, 0001, \ldots\}$

Regular?



But for $0^n 1^n$, we can't define a DFA

To accept a lang, if it is reqd to count more than 1 items DFA can't do.

Eg: A Regular Language

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

$= \{0101, 1010, 1111, 11001, \ldots\}$

The DFA:

→ has 5 states, named 0, 1, ..., 4

→ correspond to 5 remainders of an integer division by 5

→ start and only final state is 0

→ If string w represents an integer i then assume

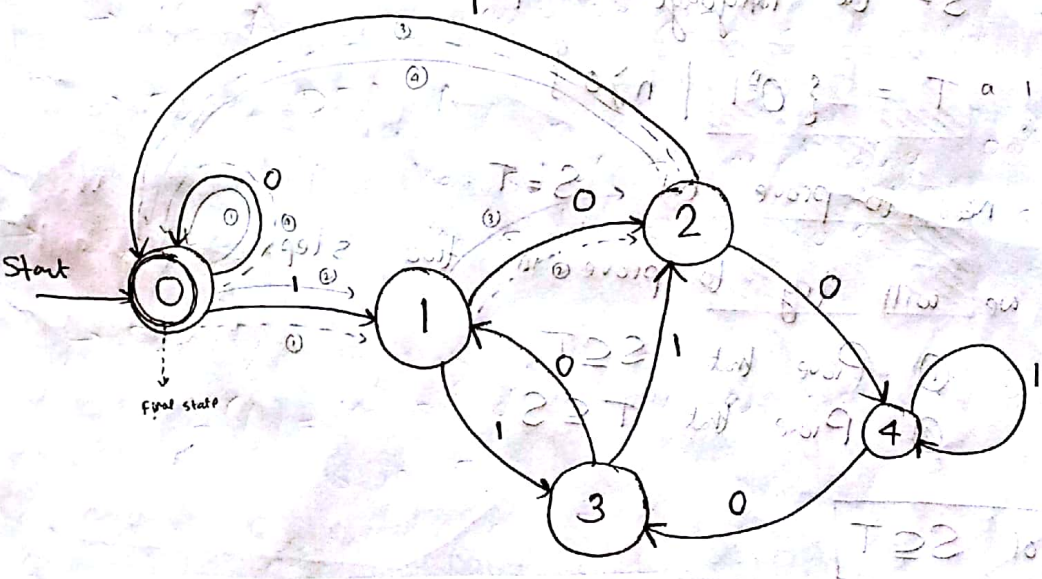
$$\delta(0, w) = i \% 5$$

→ w0 represents $2i$

So we want $\delta(i\%5, 0) = 2i \% 5$

→ w1 represents $2i+1$

So $\delta(i\%5, 1) = (2i+1)$



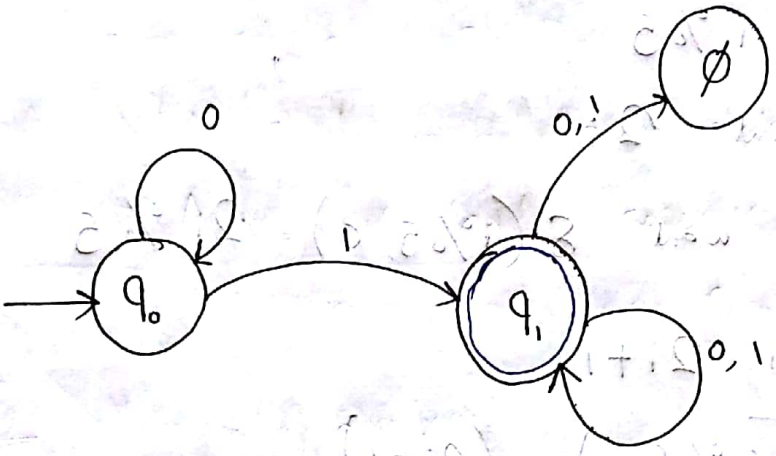
a 0 after state
1 ≡ 10
↓
2×1 = 2

a 1 after 1
11 ⇒ 2×1+1
= 3

09/08/18

A DFA accepting all binary strings starting with no, one or more 0's followed by a single 1

Eg: $\{1, 01, 001, 0001, \ldots\}$ —①



(mathematically prove)
We hv to show that ① is equal to the set, which this DFA accepts. So we need to prove the former is a subset of latter & vice-versa

Let S = the language accepted by the DFA

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

We need to prove that $S = T$.

This, we will try to prove in two steps

① Prove that: $S \subseteq T$

② Prove that $T \subseteq S$,

Proof of $S \subseteq T$

Induction Hypothesis (IH):

Any string x with $|x| \leq n$ accepted by the DFA is in T, where n is a positive integer.

As per the IH, if $\delta(q_0, x) = q_f$ then x is in T.

Consider a string ax in S where a = a single symbol, 0 or 1

Then $\delta(q_0, ax) = \delta(\delta(q_0, a), x)$

where, δ = extended transition func^n of the DFA

x can be accepted by the DFA if

$\delta(q_0, a) = q_0$

If $\delta(q_0, a) = q_0$, then a must be 0.

If $a = 0$, then $\delta(q_0, 0x) = \delta(\delta(q_0, 0), x)$

$= \delta(q_0 x) = q_f$

So, all strings accepted by this DFA, belongs to the set T

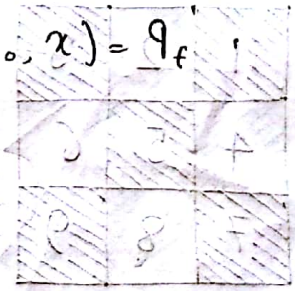
Hence, $S \subseteq T$.

Proof of $T \subseteq S$

IH:

For $x \in T$ and $|x| < n$,

if $x \in T$ then $\delta(q_0, x) = q_f$



Consider $ax \in T$

Since $|ax| = n > 1$, $a = 0$

For $a = 0$, $\delta(q_0, 0x) = \delta(\delta(q_0, 0), x)$

$= \delta(q_0, x) = q_f$

So, for any string accepted by the DFA, the string is in T.

(ie. all strings $\in T$ will be accepted by the DFA)

Hence $T \subseteq S$

Finally, we prove $S = T$.

NON - DETERMINISM

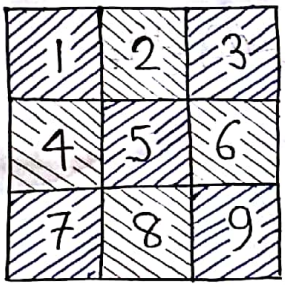
→ A Nondeterministic Finite Automata (NFA) has the ability to be in several states at once

→ Transition from a state on an input symbol can be to some set of states.

↳ An NFA starts in one start state.

→ Accepts if any sequence of choices of state input pairs leads to a final state

Eg: Moves on a chessboard.



→ States = Squares

→ Inputs = r (move to an adjacent black square) and b (move to an adjacent blue square)

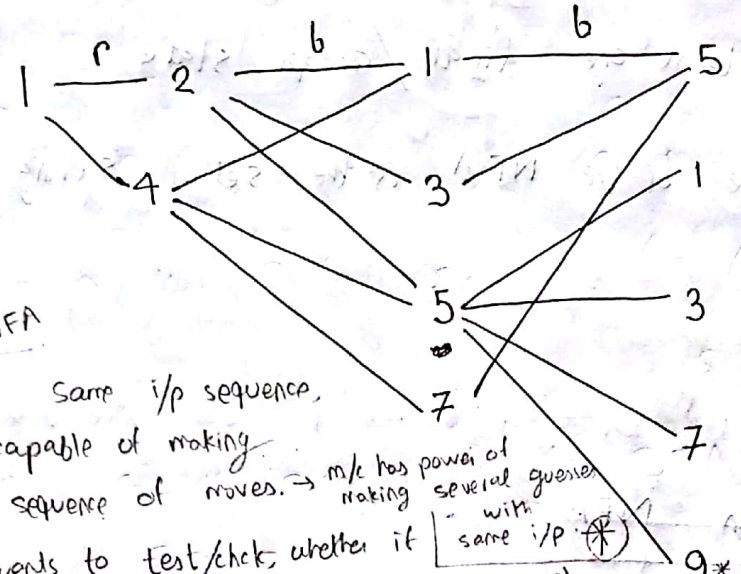
→ Start state, Final state are in opposite corners. (So if start state = 1, opp state = 9)

State	r	b
→ 1	2, 4	5
2	4, 6	1, 3, 5
3	2, 6	5
4	2, 8	1, 5, 7

State	r	b
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* (final state)	6, 8	5

For any state-i/p pair, the next state may not be unique (it will have a next set of states)

Eg: input string = r b b



If m/c reads input symbol through this seq. of moves,

$\left(1 \xrightarrow[r]{} 4 \xrightarrow[b]{} 5 \xrightarrow[b]{} 9\right)$

we can declare that this seq. 'rbb' will be accepted by the m/c.

Adv. of NFA

→ With the same i/p sequence, it is capable of making several sequence of moves. → m/c has power of making several guesses with same i/p (*)

→ Bcos, it wants to test/check whether it with any of this seq. of states, the accept state can be reached finally

(*) which is not possible in DFA. In deterministic al there is no scope of making guess.

→ Family of lang. which can be accepted by all possible DFA, is same as that accepted by all possible NFA

→ Moreover, every NFA can be converted to an equivalent DFA

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 final states $F \subseteq Q$

→ The transition func^n δ is defined as:

$$\delta(q, a) = \text{a set of states}$$

Extended to String as follows:

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

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

⇒ Language of an NFA :

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

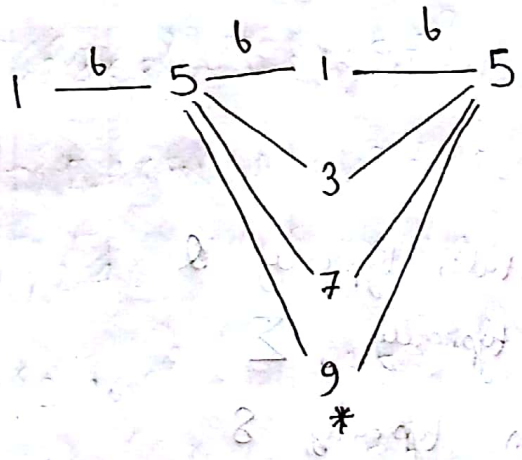
→ The language of the NFA is the set of string it accepts

If the m/c is supplied with a string consisting of even no. of 'b's, only then it will be accepted.

Eg : Language of an NFA :

→ For the chessboard NFA, we have seen that rbb is accepted

→ If the i/p consists of only b's :



It will alternate bet^n 1, 3, 7, 9 and 5.

If this m/c is supplied with even no. of b's only, then that will be accepted

NOTE For a lang $0^n, 1^n$, no DFA can be formed (proved earlier)
For this lang no NFA can't be formed
A lang which is not accepted by a DFA, can't be accepted by a NFA

EQUIVALENCE OF DFA's, NFA's

16/8/18

→ 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 in a set containing exactly one state — the state the DFA is in after reading the same input

Every i/p string accepted by DFA will also be accepted by NFA
rejected — rejected

In case of DFA, for every state-input transition, nxt i/p is a singleton state.
To convert every state transition, we hv to get a singleton set

→ For every NFA, there is a DFA that accepts the same language.

→ Proof is the subset construction.

→ Given an NFA with states Q, input alphabet Σ, transition function δ_N, start state q_0, set of final states F, construct an equivalent DFA with:

→ states 2^Q (set of subsets of Q) [each subset of Q = state of DFA]
all possible subsets of Q = sets of DFA
Q = set of states of the NFA

→ Input alphabet Σ (∵ DFA will accept same language accepted by NFA)

→ Start state = $\{q_0\}$ (this singleton set = start set of the equivalent DFA)

→ Set of Final states = all those with a member of F.

→ The transition function δ_D is defined as:

$$\delta_D(\{q_1, \ldots, q_k\}, a)$$ ← i/p symbol a
internal state of DFA

= Union over all $i = 1, \ldots, k$ of $\delta_N(q_i, a)$

$$= \delta_N(q_1, a) \cup \delta_N(q_2, a) \cup \ldots \cup \delta_N(q_k, a)$$

certain subsets will contain some of more final states of NFA. Those subsets will be considered final state of the DFA

So DFA may hv 1/more final states depending on: at least one final state of NFA

NFA → equivalent DFA

(Q) (2^Q)

$= \{q_1, \ldots, q_k\} \Rightarrow$ a typical state of the DFA

All states in 2^Q may not be reachable from strt state of DFA.

Eg: Only for those states which are reachable from strt state of DFA we will define δ_D. No need for finding state transitions for the rest.

Let us construct the DFA equivalent to our chessboard NFA.

(To find equivalent DFA from a NFA)

→ **Subset Construction** :

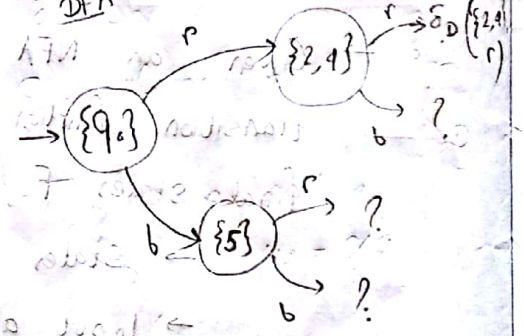
State	r	b
→ {1}	{2,4}	{5}
{2,4}	{2,4,6,8} ①	{1,3,5,7} ②
{5}	{2,4,6,8}	{1,3,7,9} ③
{2,4,6,8}	{2,4,6,8}	{1,3,5,7,9}
{1,3,5,7}	{2,4,6,8}	{1,3,5,7,9}
*{1,3,7,9}	{2,4,6,8}	{5}
*{1,3,5,7,9}	{2,4,6,8}	{1,3,5,7}

①②③ are the strt. state reachable from the strt. state. So we need to find states corresponding to these also.

$\delta_D(\{1\}, r) = \delta_N(1, r) = \{2, 4\}$

$\delta_D(\{1\}, b) = \delta_N(1, b) = \{5\}$

DFA



We should find these to complete construction of this DFA

$\delta_D(\{2,4\}, r) = \delta_N(2, r) \cup \delta_N(4, r)$

$\delta_D(\{2,4\}, b) = \delta_N(2, b) \cup \delta_N(4, b)$

$\delta_D(\{2,4,6,8\}, r)$

$= \delta_N(2,r) \cup \delta_N(4,r) \cup \delta_N(6,r) \cup \delta_N(8,r)$

$= \{4,6\} \cup \{2,8\} \cup \{2,8\} \cup \{4,6\}$

exclusively defined states

No such (state left which is reachable from strt state) but not in state-table.

For those states, not reachable, we might not define transition func'n.

Proof of Equivalence : Subset Construction

23/8/18

→ Let us show by induction on $|\omega|$ such that

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

Basis : $\omega = \epsilon$;

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

Induction Hypothesis : Assume for strings shorter than $|\omega|$ the above hypothesis is true.

Let $\omega = xa$, IH holds for x ($\because$ length of $x < |\omega|$)

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

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

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

→ For NFA : the extension of δ_N

→ For DFA : Definition of δ_D plus extension of δ_D.

→ That is, $\delta_D(S, a) = T$ then extend δ_D to $\omega = xa$

$$\delta_N(q_0, xa) = \delta_N(\underbrace{\delta_N(q_0, x)}_{S}, a) \quad \Big| \quad \delta_D(\{q_0\}, xa) = \delta_D(\underbrace{\delta_D(\{q_0\}, x)}_{S}, a)$$

$S = \{P_1, P_2, \ldots\}$

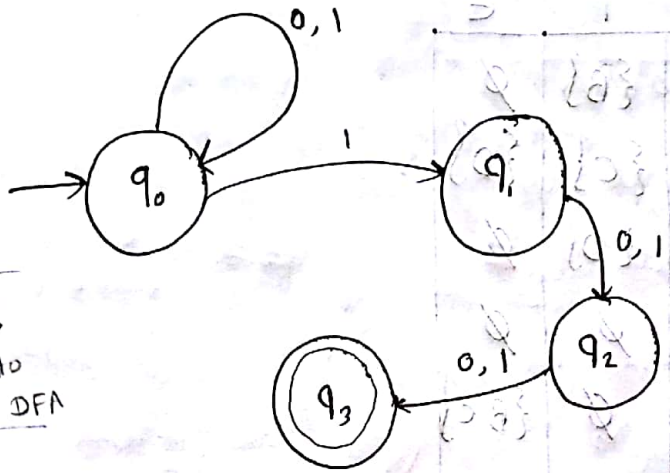
Q Give a DFA accepting the language L which consists of all strings over $\{0,1\}$ containing a 1 in the 3rd pos^n from the end.

A. Eg: 000100, 0101100 → strings of the language

0111011 X → Not a string of the language

Direct DFA → difficult to design

NFA → m/c allows to make some guess & verify it → thus easier to design NFA



NFA:

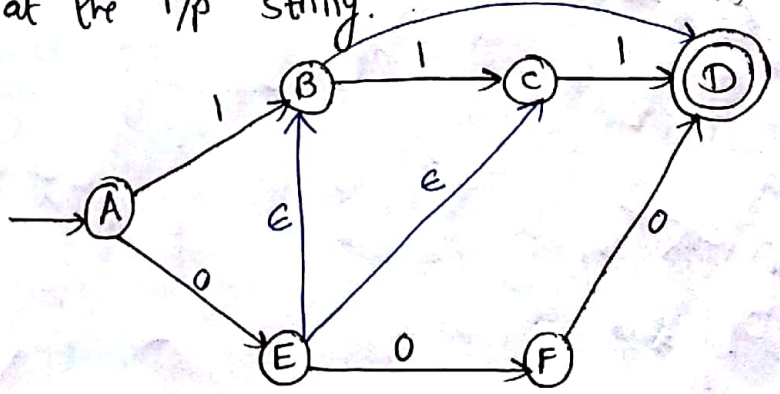
accepting all bin. strings in which a 1 occurs in 3rd pos^n from end.

Now it can be easily convated into equivalent DFA

NFA's with ϵ-transitions

→ We can allow state to state transitions on ϵ input.

→ These transitions are done spontaneously, without looking at the i/p string.



NFAϵ

Eg. of an NFA with ϵ-transition

$\delta(A, 0) = ?$

$= \{E, B, D, C\}$

TRANSITION TABLE

State	0	1	∈
→ A	{E}	{B}	∅
B	∅	{C}	{D}
C	∅	{D}	∅
D*	∅	∅	∅
E	{F}	∅	{B, C}
F	{D}	∅	∅

Closure of states $(Cl(q))$

= Set of states you can reach from q following only arcs labelled ∈

Eg: $\begin{cases} Cl(A) = \{A\} \\ Cl(E) = \{E, B, C, D\} \end{cases}$

Closure of a set of states:

= Union of the closure of each state

Extended Delta (δ) for NFA_ϵ's

24/8/18

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

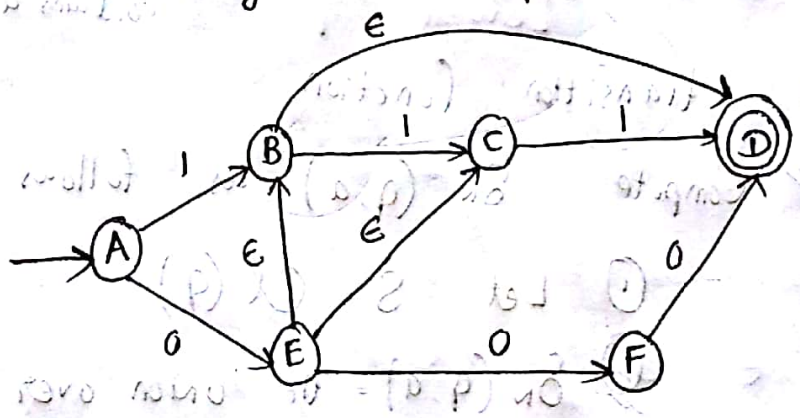
→ Induction: $\hat{\delta}(q, xa)$ is computed as follows:

① Start with $\hat{\delta}(q, x) = S$

② Take the union of $Cl(\delta(p, a))$ for all p in S

Intuition:

$\hat{\delta}(q, w)$ = the set of states you can reach from q following a path labeled w



Eg.: Extended Delta ..

→ $\hat{\delta}(A, \epsilon) = Cl(A) = \{A\}$

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

→ $\hat{\delta}(A, 01) = \hat{\delta}(B, 1) \cup \hat{\delta}(C, 1) \cup \hat{\delta}(D, 1) \cup \hat{\delta}(E, 1)$

$\hat{\delta}(\hat{\delta}(A, 0), 1)$ ($\hat{\delta}(A, 0) = Cl(E)$)

$= Cl\{C, D\}$

$= \{C, D\}$

EQUIVALENCE OF NFA, NFA_ϵ :

an NFA that accepts the same language.

→ We do so by combining ϵ-transitions with the next transition on a real input

→ Start with an NFA_ϵ with

Q = the set of states

Σ = the i/p alphabet

q_0 = the start state in Q

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

δ_N = the transition function

Compute $\delta_N(q,a)$ as follows:

① Let $S = cl(q)$

② $\delta_N(q,a)$ = the union over all p in S of $\delta_\epsilon(p,a)$

INTUTION:

δ_N incorporates ϵ-transitions before using 'a', but not after.

Eg: ϵ-NFA to NFA

δ_N

State	Input 0	1
→ A	{E}	{B}
B	∅	{C}
C	∅	{D}
D	∅	∅
E	{F}	{C, D}
F	{D}	∅

$\delta_N(B,0) = \delta_N(cl(B),0)$

$\delta_N(C,0) = \delta_N(cl(C),0) = \emptyset$

$\delta_N(C,1) = \{D\}$

First find closure. Then find $\delta_N(\ ,0)$ $\delta_N(\ ,1)$

∴ They are equivalent.

Regular Expression (REs)

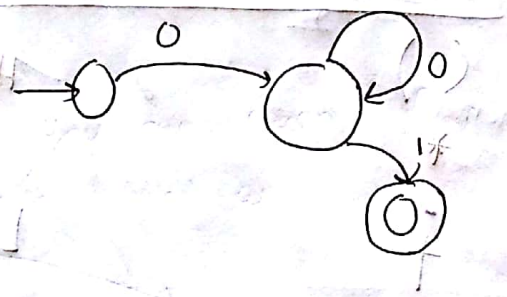
→ REs are an algebraic way to describe languages.

→ They describe exactly the regular languages

→ Let $E = aR+E$

then $L(E)$ = the language E defines

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



REs : Definition :

Basis 1:

If a is any symbol, then a is a R.E. and

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

Basis 2:

ϵ is a RE and $L(E) = \{\epsilon\}$

Basis 3:

$\emptyset$ is a RE and $L(\emptyset) = \emptyset = \{\}$ $\quad \emptyset$ = empty set

Induction 1:

If E_1 and E_2 are REs then $E_1 + E_2$ is a RE

and $L(E_1 + E_2) = L(E_1) \cup L(E_2)$

Induction 2:

If E_1 and E_2 are REs then $E_1 . E_2$ is a RE and

$L(E_1 . E_2) = L(E_1) . L(E_2)$

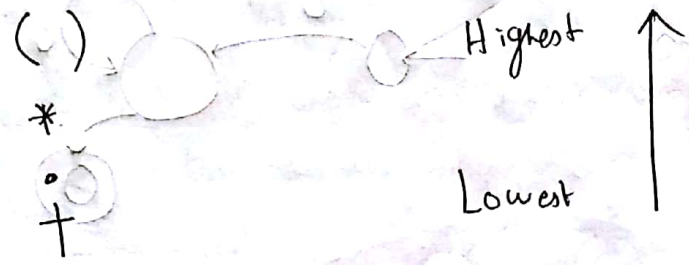
→ concatenation : The set of strings $w.x$ s.t. w is in $L(E_1)$ and x is in $L(E_2)$

Induction 3:

If E is a RE, then E^* is a RE, then

$$L(E^*) = (L(E))^* = \{\epsilon\} \cup \{L(E)\}^+$$

PRECEDENCE OF OPERATORS



Eg: $L(E) = \{0\}$

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

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

$L(E^*) = \{\epsilon\} \cup \{$ all possible binary strings $\}$

Eg: RE's

$L(\bar{0}, \bar{1}) = \{01\}$ $\quad \because L(\bar{0}) = \{0\}, \; L(\bar{1}) = \{1\}$

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

$L(\bar{0}(\bar{1} + \bar{0})) = L(\bar{0}) . L(\bar{1} + \bar{0})$

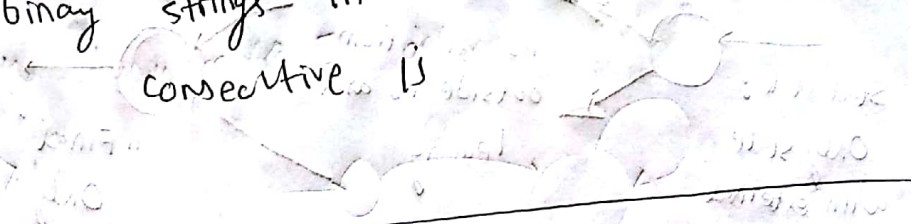
$= \{0\} . \{0, 1\} = \{00, 01\}$

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

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

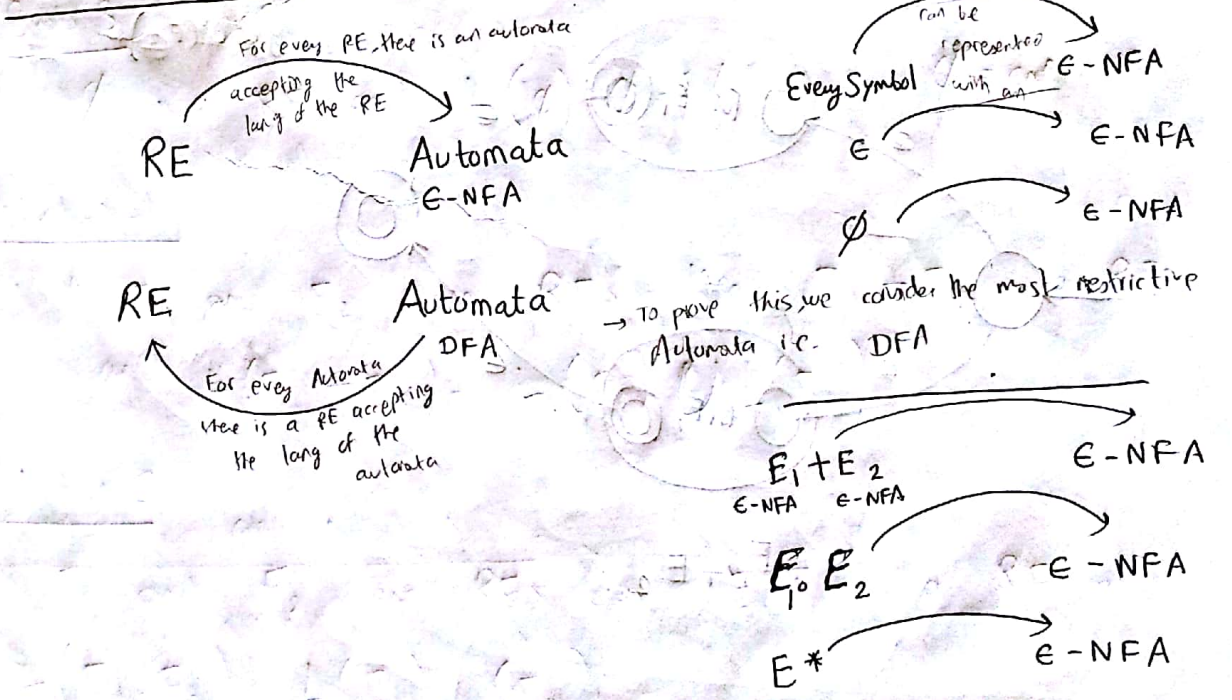
↑ all binary string no consecutive 1's every string ends with 0

= all binary strings in which there are no consecutive 1's



29/8/18

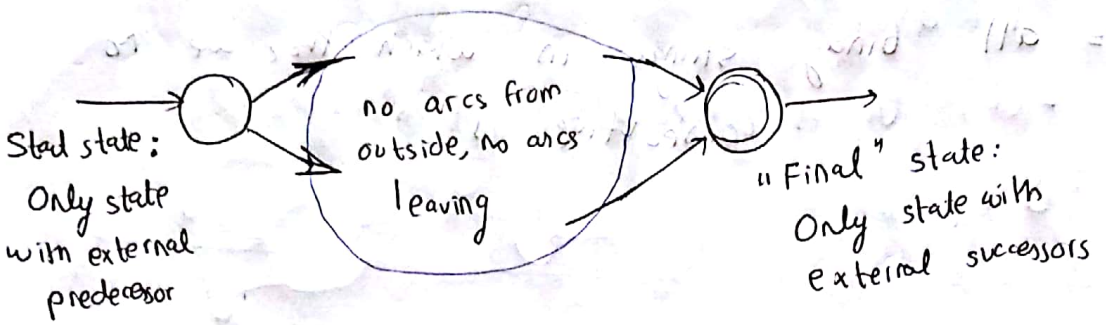
EQUIVALENCE OF REGULAR EXPRESSIONS & AUTOMATA



→ RE to ϵ-NFA: Basis

→ Symbol a :

→ We always construct an automaton of a special form as follows:



→ ϵ

If ϵ is a reg. expression,

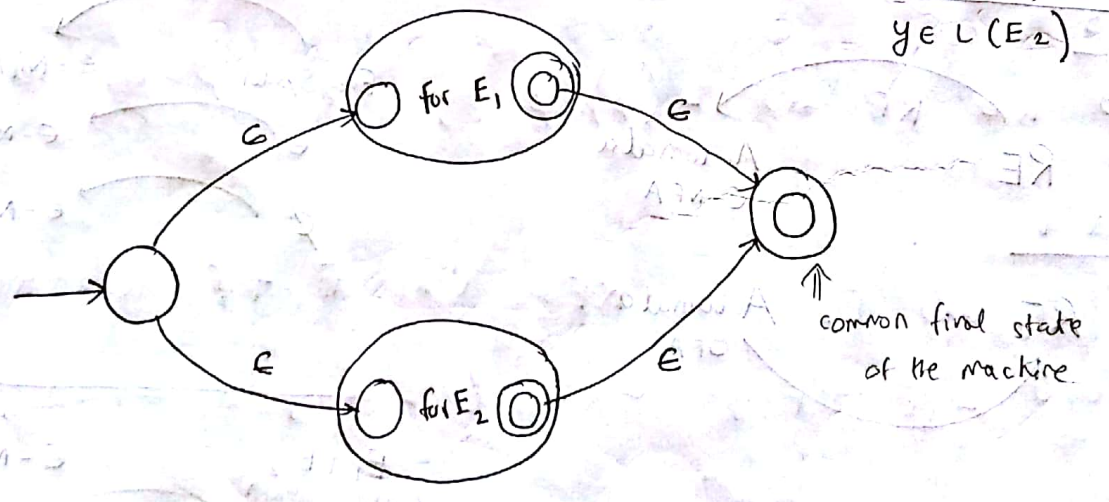
Then this is the automata accepting the regular expression

→ ∅

→ RE to ϵ-NFA: Induction 1 - Union

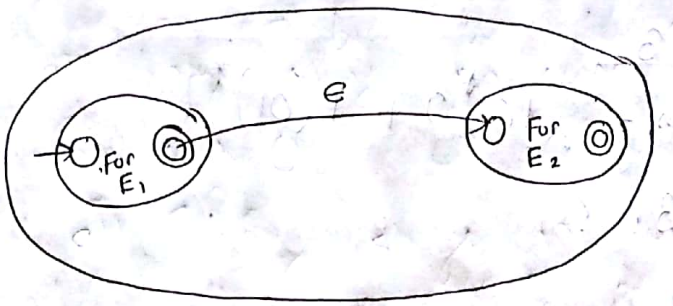
$x \in L(E_1)$

$y \in L(E_2)$



$E_1 + E_2$

→ RE to ϵ-NFA: Induction 2 - Concatenation



For $E_1 E_2$

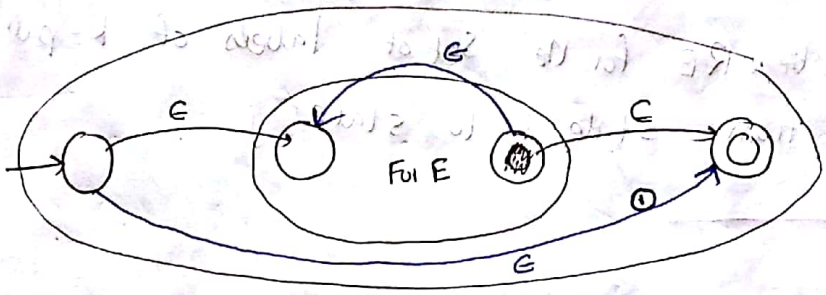
lang. of E_1 concatenated E_2

$$L(E_1 . E_2) = L(E_1).L(E_2)$$

Every string in $L(E_1 . E_2)$ will be of the form xy such that x in $L(E_1)$ & y in $L(E_2)$

i.e. concatenation operation is not commutative

→ RE to ϵ-NFA: Induction 3 - Closure



For E^*

without ①, it accepts any of E^*, but not guaranteed for ϵ

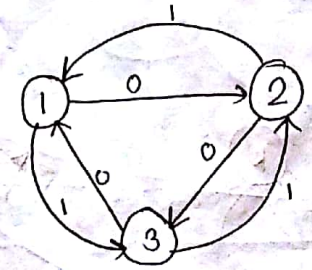
✓ DFA to RE

→ 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 / states are not restricted; they can be any state

Eg: k-Paths



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

If we can find 3-path betⁿ 2 to 3, then we will be able to find all paths betⁿ 2 to 3

→ **K-path Induction:**

Let R_{ij}^{*} = the RE for the set of labels of k-paths from state i to state j

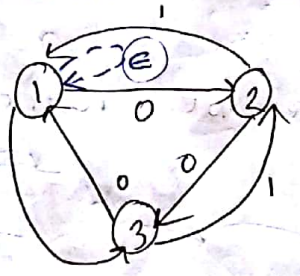
→ **Basis:**

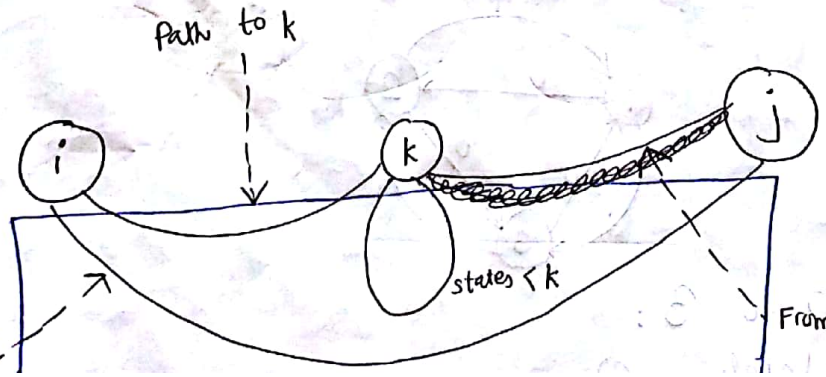
$k=0$, R_{ij}^{0} = sum of labels of arcs from i to j.

$\emptyset$ if no such arc $R_{12}^{0} = 0$

But add ϵ if $i=j$ $R_{11}^{0} = \emptyset + \epsilon$

If R_{11}^{0} we can think of a path like: ϵ





→ K-Path: Inductive Case

→ A k-path from i to j either:

① Never goes through state k

② Goes through k one or more times

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

- R_{ij}^{k-1}: Does not go through k
- R_{ik}^{k-1}: Goes from i to k the first time
- $\left(R_{kk}^{k-1}\right)^{*}$: zero or more times from k to k
- R_{kj}^{k-1}: Then from k to j

→ Final Step:

→ The RE with the same language as the DFA is the sum (union) of R_{ij}^{n}, where:

① n is the no. of states, i.e. paths are unconstrained

② i is start state

③ j is one of the final states

31/8/18

A LANGUAGE CLASS

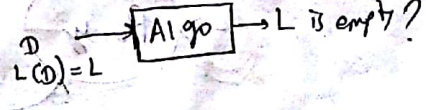
→ It is a set of languages

→ We have one example: the regular languages

→ We will see many more in this class.

TWO IMPORTANT PROPERTIES OF LANGUAGE CLASSES

- **Decision Propedes.**



→ A Decision property for a class of languages is an Algorithm that takes a formal description of a language (e.g., a DFA) and tells whether or not some property holds.

① emptiness problem ⇓ whether empty or not

② membership problem ⇓ defines whether a string is a member of the lang - or not!

Eg: Is language L empty?

Eg: ①&②

- **Closure Properties**

→ A closure property of a language says that given languages in the class, an operator (e.g. UNION) produces another language in the same class.

Eg:

The regular languages are obviously closed under union, concatenation, and (Kleen) closure

→ Use the regular expression representation of languages.

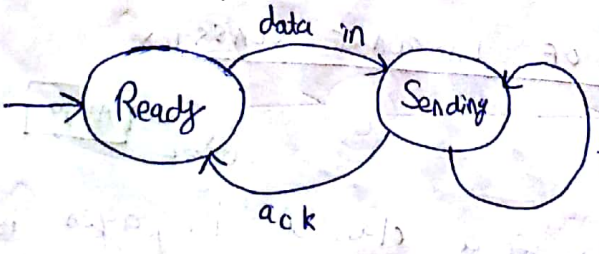
WHY DECISION PROPERTIES?

→ When we talk of protocols represented as DFA's we note that important properties of a good protocol are related to the language of the DFA

→ Eg:

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

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



Protocol for sending data

→ We might want a "smallest" representation for a language,

eg: a minimum-state DFA or a shortest RE

→ If you cannot decide "Are these two languages the same?", i.e. do two DFA's define the same language?, you cannot find a "smallest".

WHY CLOSURE PROPERTY?

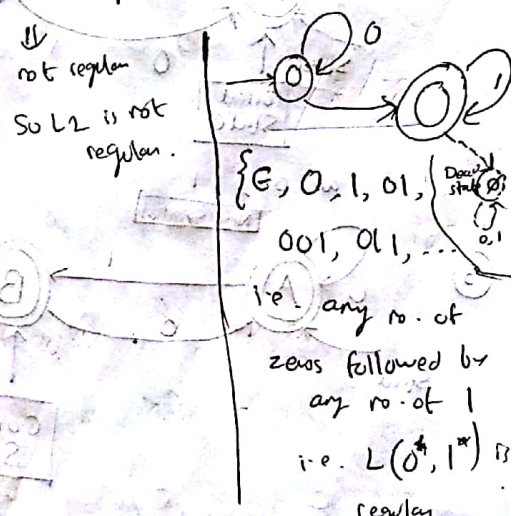
1) Helps construct representation
2) Helps show (informally described) languages not to be in the class.

Eg: Show that L_2 = the set of strings with an equal no. of 0's and 1's is not regular

$$L_2 \cap L(0^*1^*) = \{0^n 1^n \mid n \geq 0\}$$

⇓ not regular

So L_2 is not regular.



{ε, 0, 1, 01, 001, 011, ...}

i.e. any no. of zeros followed by any no. of 1

i.e. $L(0^*, 1^*)$ is regular

5/9/18

The Membership Question – A Decision Property

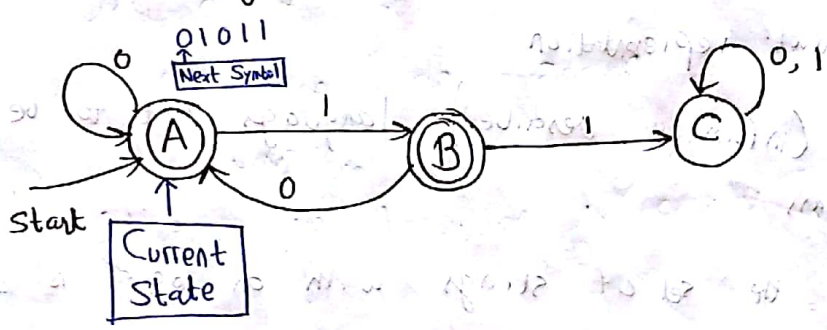
"Is string w in regular language L?"

→ Assume L is represented by a DFA A.

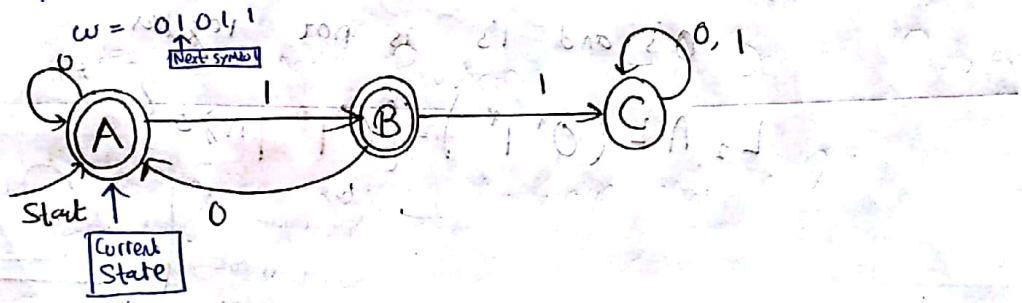
→ Simulate the action of A on the sequence of input symbols forming w.

Eg: Testing Membership.

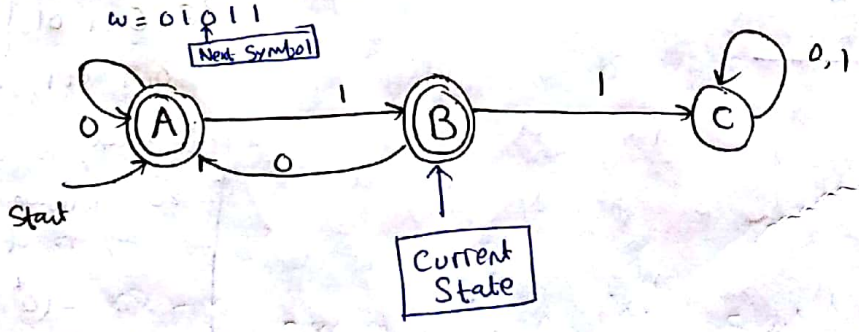
(1)



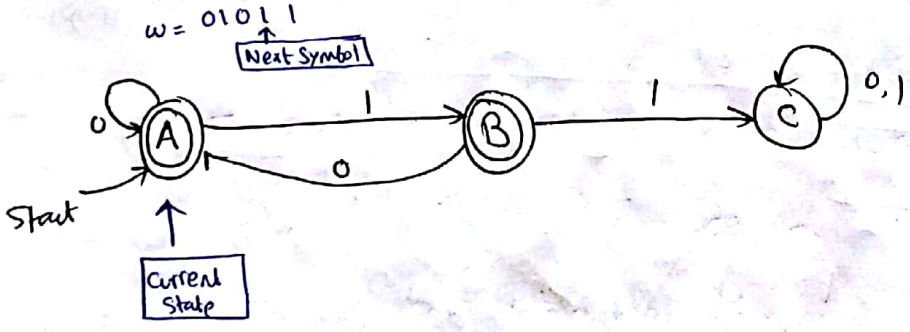
(2)



(3)

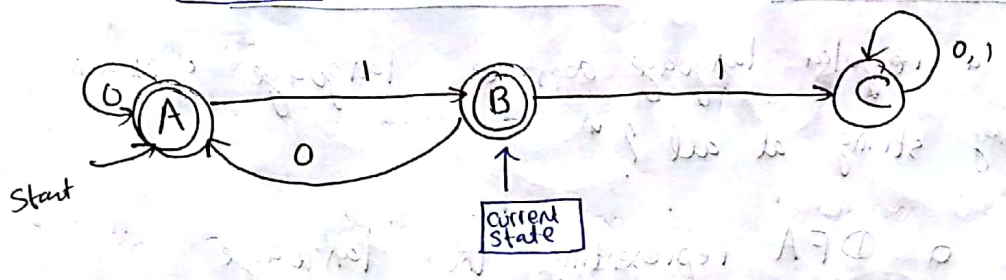


(4)



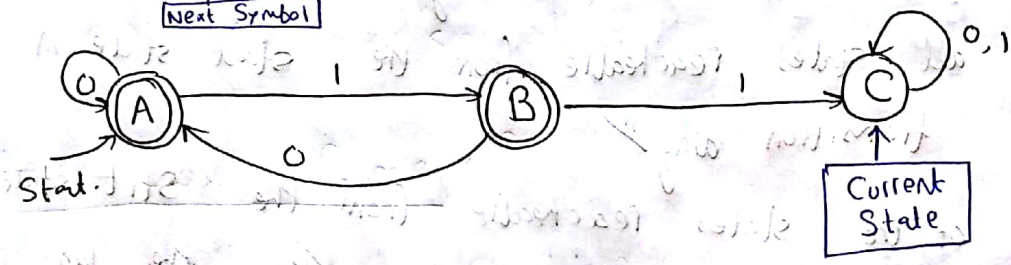
(5)

w = 01011
↑ Next Symbol



(6)

w = 01011 ∈
↑ Next Symbol



∴ Since current state is not an accept state,

$01011 \notin L$ (Language)

The string does not belong to the language.

OR

⊢ ≡ a move of the m/c.

$\delta(A, 01011) \vdash_A \delta(A, 1011)$

$\vdash_A \delta(B, 011)$

$\vdash_A \delta(A, 11)$

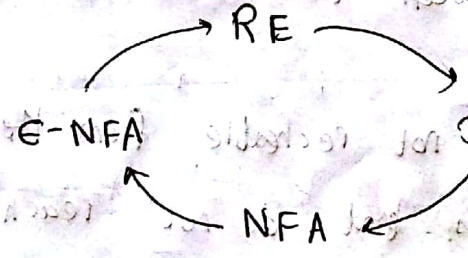
$\vdash_A \delta(B, 1)$

$\vdash_A \delta(C, \epsilon)$ Not accept.

∴ $01011 \notin L$

Membership prob for a regular lang is decidable (bcos we have an algo to find a soln to whether it belongs to lang)

→ What if the RL is not represented by a DFA?



→ The Emptiness Problem – Another Decision Problem

"Given a regular language, does the language contain any string at all?"

① Assume a DFA representing the language
② Draw / Form the transition diagram of the DFA
③ Find all states reacheable from the start state in the transition diagram
④ Out of the states reacheable from the strt state, if any one is an accept state, then the lang is not empty.
Otherwise, the language is empty.

→ The Infiniteness Problem – A Decision Problem

"Is a given regular language infinite?"

we need to find a cycle b/w strt state & final state
If present, the lang can accept all strings

First eliminate all the states which are not reacheable from strt state.
Next eliminate all states from which accept state is not recheable.
If no cycle bet^n strt state & any of the accept state of the m/c we can say it is not finite.

① Find cycles between the start state & the final state
useless in this context
② Eliminate states not recheable from the strt state
③ Eliminate states that do not reach a final state
④ Test if the remaining transition graph has any cycles.

→ PUMPING LEMMA FOR RL's :

→ For every regular language L, there is an integer n such that for every string w in L of length $\geq n$ we can write $w = xyz$ such that

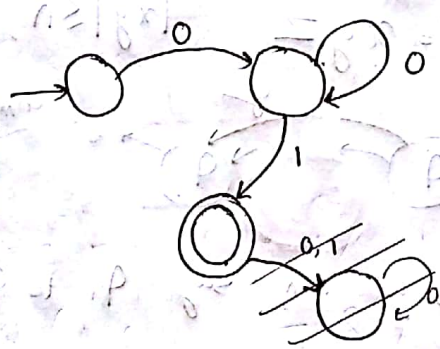
n = no. of states in the DFA of the lang.

① $|xy| \leq n$

② $|y| > 0$

③ For all $i \geq 0$, xy^iz is in L.

$\{0^k1 \mid k \geq 1\}$ $\{01, 001, 0001, \ldots\}$



n = no. of states = 3

pump $y = 0$

$|w| = |\underbrace{0}_{x}\underbrace{0}_{y}\underbrace{1}_{z}| \geq 3$

$|xy| \leq 3$

$|y| > 0$

• ⇒ 0's produced due to pumping action.

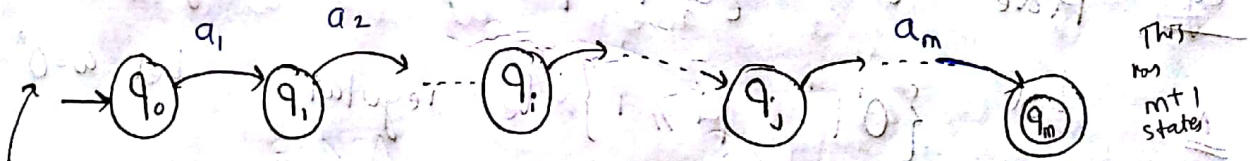
If it is pumped once : 0001

twice : 00001

PROOF:

$|w| \geq n$, $w \in L$ (w is a string from the lang. whose length is at least n)

n = no. of states of the DFA of L

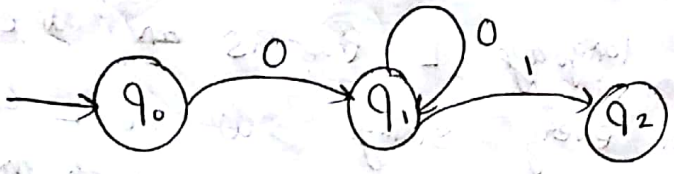


The sequence of transitions the DFA has on receiving w as input is

Let $w = a_1a_2 \ldots a_m$ where $m \geq n$

Supposing $m = n$, the seq. has $n+1$ states. But the DFA has

n states. So 2 states are equal.



$w = 0001$

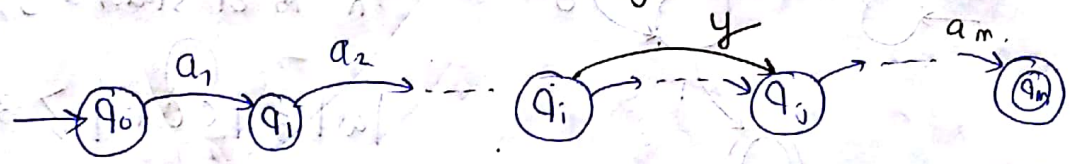
$q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_1 \xrightarrow{0} q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_2$

We name the states like this as we don't know any states. ⇒ q_0 q_1 q_2 q_3 q_4 q_5

From $q_0 - q_5$ ⇒ No. of states ≥ 3

So at least 2 states are identical.

Let $q_i = q_j$ $\quad w = xyz$, $|xy| \leq n$, $|y| > 0$



NOTE: q_j is the 1st repitition of ($q_i = q_j$)

So for prev eg. $q_1 \rightarrow q_2$ can be said as y

$q_i = q_j$, So we can say q_i to q_j is 'y'.

$\Rightarrow xy^i z \in L$ $\quad i = 0, 1, 2, \ldots$

Hence the Pumping Lemma is proved

Do simulate probs

Eg: Prove that $\{0^i 1^i \mid i \geq 1\}$ is not regular.

A. Let $\{0^i 1^i \mid i \geq 1\}$ be regular. Let $w = 0^n 1^n$

According to pumping lemma (P.L.), let us choose w in L where $|w| \geq n$; n = the const of P.L.

According to P.L.,

$w = xyz$

such that $|xy| \leq n$ & $|y| > 0$. and $xy^i z \in L$.

xy solely consists of 0's.

The string xz will be of the form $0^{n-k}1^n$ where $k = |y|$.

$\therefore 0^{n-k}1^n \notin L$ $\therefore$ The language is not regular (by P.L.)

Equivalence: A Decision Property — 6/9/18

→ Given regular languages L and M, is $L = M$?

→ Product DFA from DFA's for L and M

Let the DFA's for L and M have sets of states Q and 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)

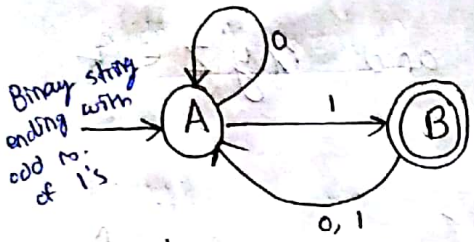
Transitions:

$$\delta([q,r], a) = [\delta_L(q,a), \delta_M(r,a)]$$

where δ_L, δ_M are the transition functions for DFA's L, M.

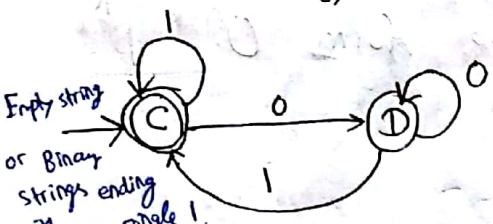
That is, we simulate the two DFA's in the two state components of the product DFA.

Eg: Product DFA



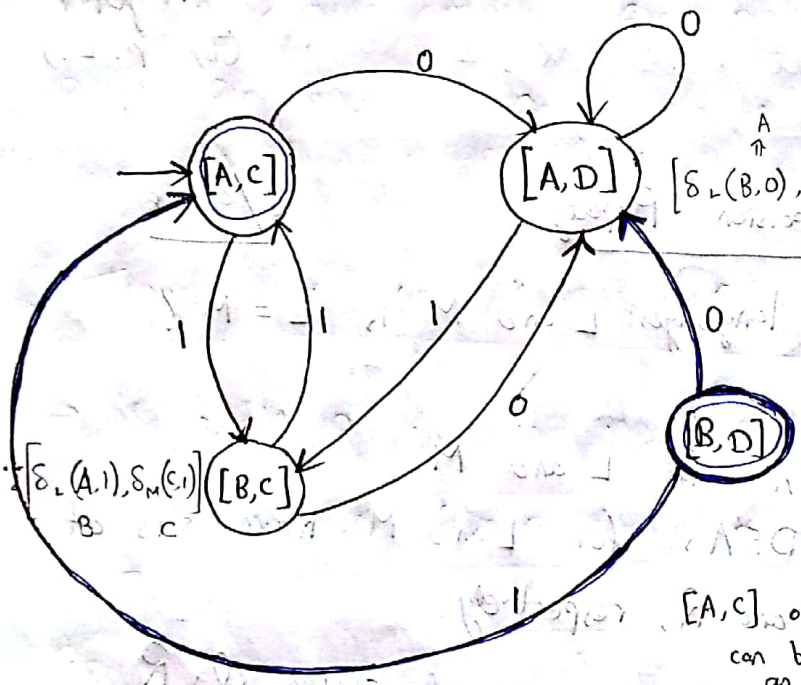
$Q = \{A, B\}$

DFA accepting all bin. strings ~~odd no: of~~ 1 occurs at end.



$R = \{C, D\}$

DFA that accepts all binary strings ending with a 1



$[\delta_L(B,0), \delta_M(C,0)]$

$[\delta_L(A,1), \delta_M(C,1)]$

[A,C] or [B,D] can be marked as accept state.

• ⇒ Though present in m/c, but not reacheable from strt-state. (as all paths from this state are outgoing) So no meaning in showing this.

Equivalence Algorithm :

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

→ Thus the product accepts w iff w is in exactly one of L and M.

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

→ We already have an algorithm to test whether the language of a DFA is empty.

CONTAINMENT: A DECISION PROPERTY

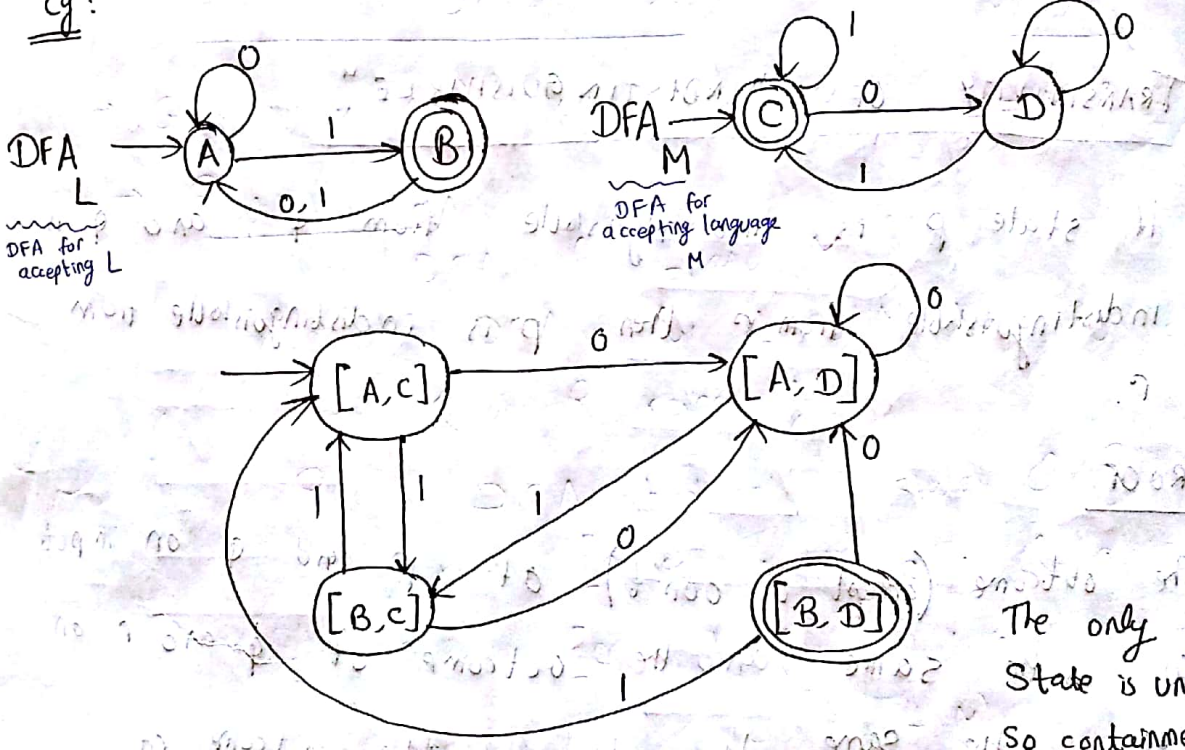
7/9/18

→ Given regular languages L and M, is $L \subseteq M$

→ form the product automata

→ Define the final states [q,r] of the product as q is Final and r is not so its language is empty iff $L \subseteq M$.

Eg:



The only Final State is unreachable So containment holds.

THE MINIMUM STATE DFA FOR A REGULAR LANGUAGE

→ Construct a table with all pairs of states

→ If you find a string that distinguishes two states (takes exactly one to an accepting state), mark that pair

→ Algorithm is a recursion on the length of the shortest distinguishing string.

State Minimization :

Basis :

Mark a pair if exactly one is a final state

Induction :

Mark $[q,r]$ if there is some input symbol, a, such that $[\delta(q,a), \delta(r,a)]$ is marked.

After no more marks are possible, the unmarked pairs are equivalent and can be merged into one state.

⇒ TRANSITIVITY OF "INDISTINGUISHABLE"

→ If state p is indistinguishable from q, and q is indistinguishable from r then p is indistinguishable from r.

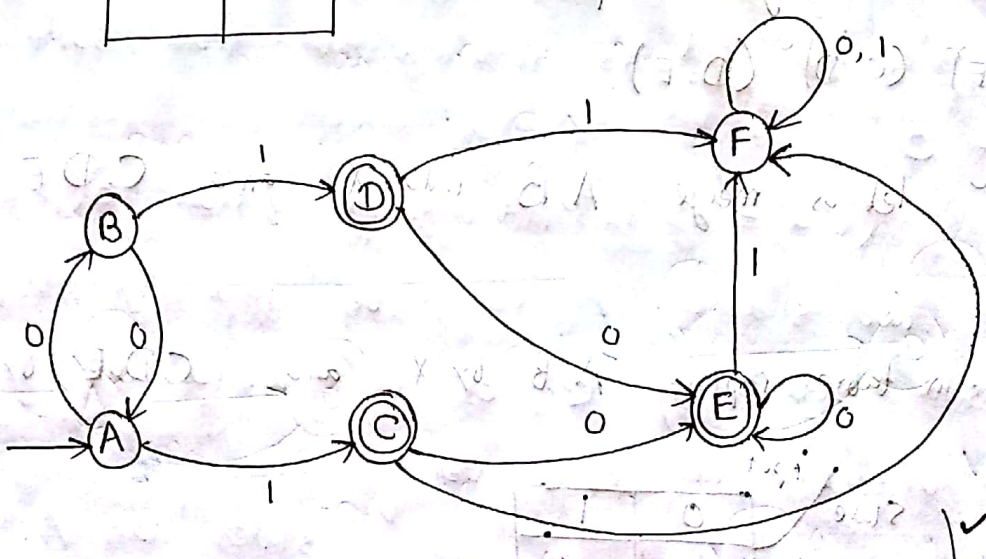
PROOF:

The outcome (accept or don't) of p and q on input w is the same and the outcome of q and r on w is the same, then likewise the outcome of p and r.

Eg:

Construct a minimum state DFA for the following.

State	Input 0	1
→ A	B	C
B	A	D
C*	E	F
D*	E	F
E*	E	F
F	F	F



✓ ⇒ distinguishable state in respect to 0-length string

✓ ⇒ 0-length
✓ ⇒ 1-length
After that no use of doing for 2-length.

	F	E*	D*	C*	B
A	✓	✓	✓	✓	•
B	✓	✓	✓	✓	•
C*	✓	•	•	•	
D*	✓	•	•		
E*	✓	•			

$[\delta(A,0), \delta(F,0)]$ (B, F) ⇒ not marked as distinguishable

$[\delta(A,1), \delta(F,1)]$ (C, F) ⇒ So [A, F] is distinguishable

$[\delta(B,0), \delta(F,0)]$ (A, F) → already marked as dist.~
So mark [B, F] as dist.

$[\delta(B,1), \delta(F,1)]$

$[\delta(A,0), \delta(B,0)]$ ⇒ not marked ⇒ leave it.
B A

$[\delta(A,1), \delta(B,1)]$ ⇒ not noted, so can't be marked this moment.
C D

Rest pairs (•) are indistinguishable.

So, the indistinguishable pairs of states are (A, B) (C, E) (C, D) (D, E).

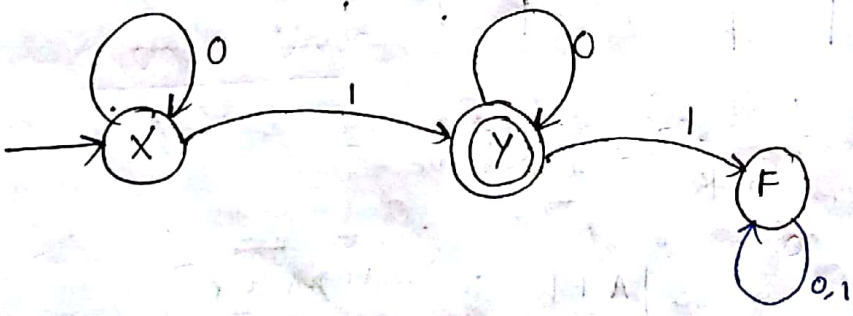
Now let us merge A, B into X and C, D, E into Y.

So in table, replace A, B by X and C, D, E by Y

State	Input 0	Input 1
X	X	Y
X	X	Y
Y*	Y	F
Y*	Y	F
Y*	Y	F
F	F	F

⇒

	0	1
X	X	Y
Y	Y	F
F	F	F



reg. exp.

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

any no. of 0 followed by 1

(or)

$= 0^*10^*$ (∵ 1st is subset of 2nd)

NOTE: Previous DFA also accept these strings. The states are minimized w/o affecting the lan.

19/9/18

PROOF:

Let A be the minimized DFA obtained by the state-minimization procedure.

Let B be a smaller equivalent.

Let us consider an automata with the states of A and B combined.

Contra positive Law

① $P \Rightarrow Q$

② $\sim Q \Rightarrow \sim P$

① = $\sim Q \vee P$

② = $P \vee \sim Q$

Distinguishability:

If for some input symbol a, $\delta(q,a)$ and $\delta(p,a)$ are indistinguishable, so are q and p

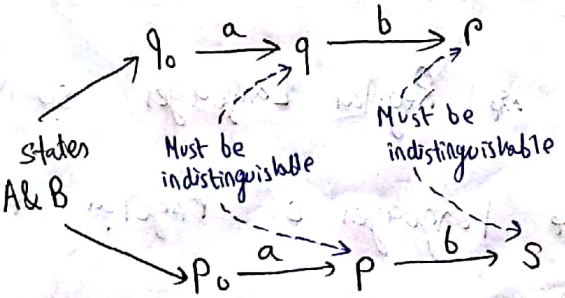
Contrapositive Form:

If states q and p are indistinguishable, so are $\delta(q,a)$ and $\delta(p,a)$

Use "distinguishable" in its contrapositive form

→ If states q & p are indistinguishable, so are $\delta(q,a)$ and $\delta(p,a)$

→ Start states of A and B are indistinguishable because $L(A) = L(B)$ (i.e. language of A and language of B are equivalent)



Inductive Hypothesis:

Basis:

Start states of A and B are indistinguishable because $L(A) = L(B)$

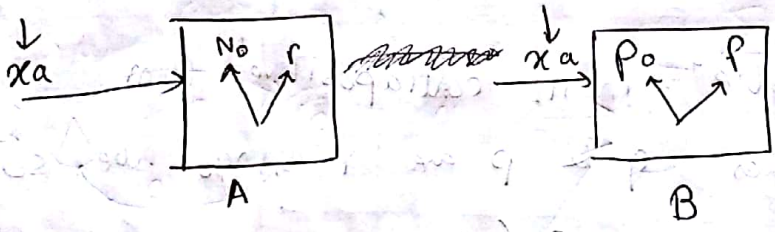
Induction:

Support $w = xa$ is a shortest string moving A to state q. By the Induction Hypothesis, x moves A to some state r that is indistinguishable from some state p of B.

Then $\delta(r, a) = q$ is indistinguishable from $\delta(p, a)$.

However, two states of A cannot be indistinguishable from the same state of B, or they would be indistinguishable from each other.

Thus, B has at least as many states as A.



CLOSURE PROPERTIES OF REGULAR LANGUAGES :

✓ Closure under Union

Prove that if L and M are Regular Languages, so is $L \cup M$

Let L and M be the languages of regular expressions R and S respectively.

Then $R + S$ is a regular expression whose language is $L \cup M$

CLOSURE UNDER INTERSECTION

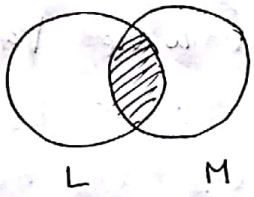
3/10/18

→ If L and M are regular languages, then so is $L \cap M$.

PROOF:

Let A and B be DFA's whose languages are L and M respectively

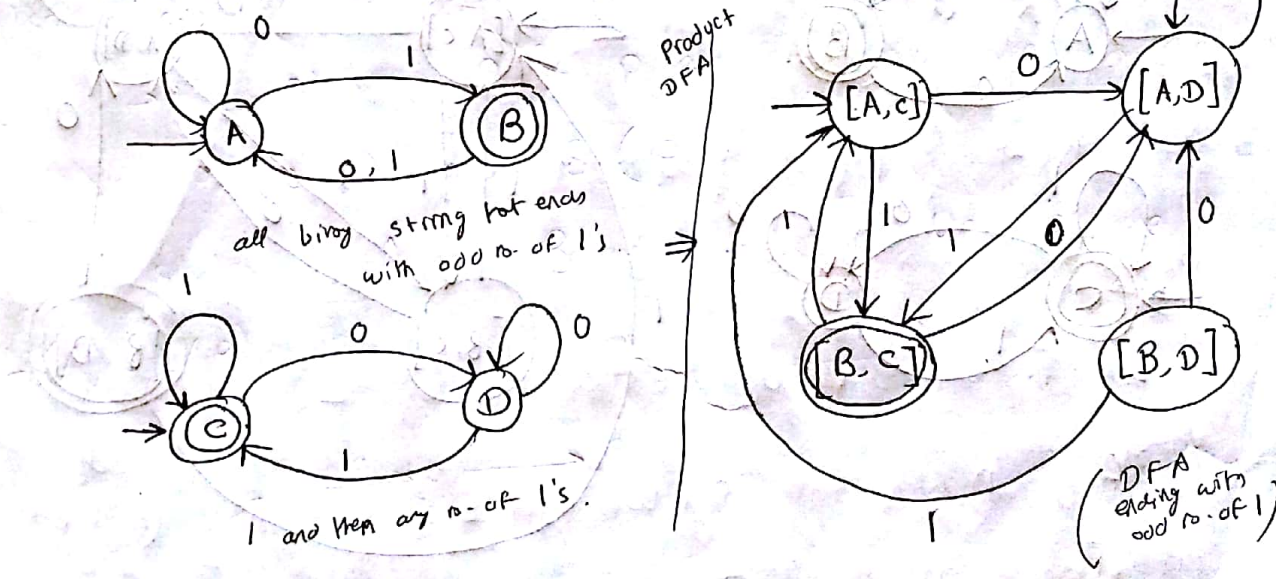
→



→ Let us construct C, the product automaton of A and B.

→ Make the final states of C be the pairs consisting of final states of both A and B.

Eg: Product DFA for Intersection.



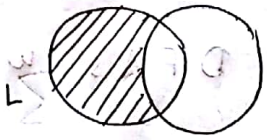
∵ B, C are final states of individual DFA's, so final state of product DFA is [B,C])

So regular lang are closed under intersection

CLOSURE UNDER DIFFERENCE:

→ If L & M are regular languages, then so is

L − M = strings in L but not M

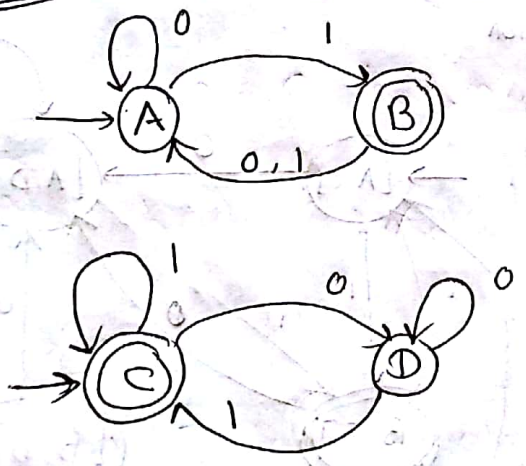


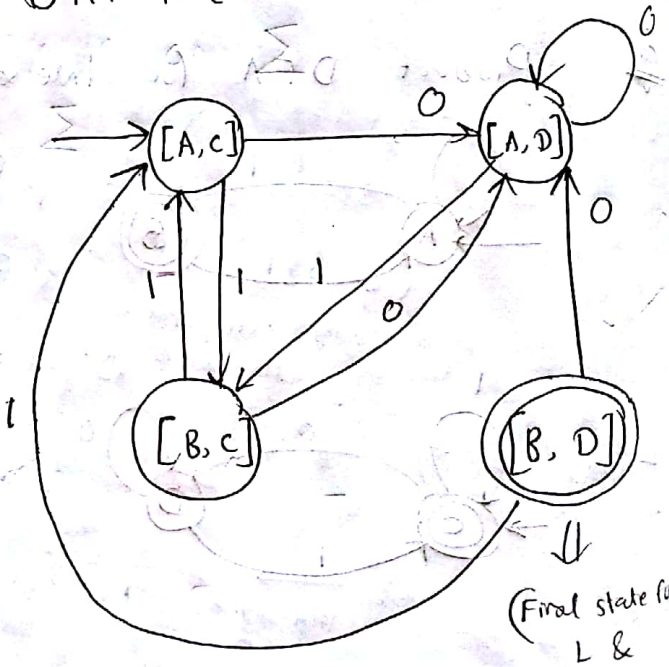
PROOF:

Let A and B be DFA's whose languages are L and M, respectively.

Let us construct C, the product automaton of A and B. Make the final states of C be the pairs where A-state is final but B-state is not.

Eg: Product DFA for Difference





(Final state for L & Non-final state for M) is the final state for (L-M).

For this particular case, [B,D] is unreacheable, so this DFA is empty. (specifying no such string which ∈ M but ∉ M)

And we again prove that DFA's are closed under difference

CLOSURE UNDER REVERSAL.

→ Given language L, L^R is the set of strings whose reversal is in L.

Eg: $L = \{0, 01, 100\}$

$L^R = \{0, 10, 001\}$

PROOF: Let E be a regular expression for L. Now we will show how to reverse E to provide a regular expression E^R for L^R.

Basis: If E is a symbol a, ϵ or $\emptyset$ then

$E^R = E$.

Induction:

If E is

→ F+G then $E^R = F^R + G^R$

→ F. G then $E^R = G^R F^R$

→ F^* then $E^R = (F^R)^*$

Eg: Reversal of a R.E.

Let $E = 01^* + 10^*$

$E^R = (01^* + 10^*)^R = (01^*)^R + (10^*)^R$

$= (1^*)^R 0^R + (0^*)^R 1^R$

$= (1^R)^* 0 + (0^R)^* 1$.

$= 1^* 0 + 0^* 1$

So family of RE are closed under reversal.

4/10/18

HOMOMORPHISM :

A homomorphism on an alphabet is a func^{n} that gives a string for each symbol in the alphabet

Eg: Let $\Sigma = \{0,1\}$ $h(0) = ab$; $h(1) = \epsilon$

→ Extended to strings by $h(a_1 \dots a_n) = h(a_1) \dots h(a_n)$

Eg: $h(0101.0) = h(0)\, h(1)\, h(0)\, h(1)\, h(0)$

$= abababab$

Closure under Homomorphism :

Let L = a regular language on Σ

h = a homom= on Σ

Now, $h(L) = \{ h(w) \mid w \text{ is in } L \}$

PROOF:

Let E be a regular expression for L.

Apply h to each symbol in E.

Language of the resulting R.E. in $h(L)$

Eg: Closure under Homomorphism

Let $\Sigma = \{0,1\}$ $h(0) = ab$; $h(1) = \epsilon$

Let L be the language of RE $01^* + 10^*$

The $h(L)$ is the language of RE

$$ab\,\epsilon^* + \epsilon\,(ab)^*$$

$$= ab\,\epsilon + \epsilon\,(ab)^*$$

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

Finally $L(ab)$ is contained in $L\left((ab)^*\right)$. so a RE for $h(L) = (ab)^*$

12/10/18

Context-Free Grammars (CFGs)

→ Informally speaking, a CFG is a notation for describing languages.

→ more powerful than FA or RE's, still cannot define all possible languages. Eg: $\{0^n 1^n 2^n \mid n \geq 1\}$ can't be defined by CFGs

→ useful for nested structures, e.g., parenthesis in programming languages

→ Basic idea is to use "variables" to stand for sets of strings (i.e. languages).

→ These variables are defined recursively, in terms of one another

→ Recursive rules ("productions") involve only concatenation

→ Alternate rules for a variable allow union.

Eg: CFG for $\{0^n 1^n \mid n \geq 1\}$

$S \rightarrow 01$

$S \rightarrow 0S1$

Basis: 01 is in the language

Induction: If w is in the language, then so is $0w1$

Another Eg:

$S \rightarrow 00S$

$S \rightarrow 11F$

$F \rightarrow 00F$

$F \rightarrow \epsilon$

→ $\{0^n 110^m \mid n \& m$ are even$\}$

CFG Formalism

→ Terminals = Symbols of the alphabet of the language being defined

→ Variables
= Non-terminals
= a finite set of other symbols, each of which represents a language

→ Start Symbol
= the variable whose language is the one being defined.

→ Productions
→ A production has the form
variable → string of variables & terminals.

→ Convention :
A, B, C, ... are variables
a, b, c, ... are terminals
w, x, y, z are strings of terminals only
α, β, γ, ... are strings of terminals and/or variables

Eg: Formal CFG

A Formal CFG for $\{0^n 1^n \mid n \geq 1\}$

Terminals = $\{0, 1\}$

Variables = $\{S\}$

Start Symbol = S

Productions =

$S \rightarrow 01$

$S \rightarrow 0S1$

Derivations – Intuition

→ We derive strings in the language of a CFG by starting with a start symbol, and repeatedly replacing some variable A by the right side of one of its productions

→ That is, the "productions for A" are those that have A on the left side of the →

Derivations – Formalism

→ We say $\alpha A \beta \Rightarrow \alpha \gamma \beta$ (⇒ : derives) if $A \rightarrow \gamma$ is a production

Eg:

$w = 000111$

$S \Rightarrow 0S1 \Rightarrow$

$00S11 \Rightarrow 000111$

↓ applying production S produces 01

→ Iterated Derivation

✓ $\overset{*}{\Rightarrow}$ means "zero or more derivation steps"

✓ Basis

$\alpha \overset{*}{\Rightarrow} \alpha$ for any string α

✓ Induction :

if $\alpha \overset{*}{\Rightarrow} \beta$ and $\beta \Rightarrow \gamma$, then $\alpha \overset{*}{\Rightarrow} \gamma$

Eg: Iterated Derivation

$S \to 0S1$ $S \to 01$

$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111$

So, $S \overset{*}{\Rightarrow} S$; $S \overset{*}{\Rightarrow} 0S1$; $S \overset{*}{\Rightarrow} 00S11$;

$S \overset{*}{\Rightarrow} 000111$

Sentential Forms

→ Any string of variables and/or terminals derived from the start symbol is called a sentential form

→ Formally, α is a sentential form if $S \overset{*}{\Rightarrow} \alpha$

→ Languages of a Grammar → Context-Free Grammar.

→ If G is a CFG, then L(G), the language of G, is $\{ w \mid S \overset{*}{\Rightarrow} w \}$

NOTE: w must be a terminal string. S is the start symbol

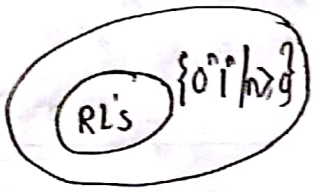
Eg: G has productions $S \to \epsilon$ and $S \to 0S1$

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

Context - Free Languages

→ A language that is defined by some CFG is called a context-free language

→ There are CFL's that are not regular languages $\{0^n 1^n 2^n \mid n \geq 0\}$



→ But not all languages are CFL's

→ Intuitively CFL's can count two things, not three

Leftmost & Rightmost Derivations : 31/10/18

→ Derivation allows us to replace any of the variables in a string

→ Leads to many different derivations of the same string

→ By forcing the leftmost variable (or alternatively, the rightmost variable) to be replaced, we avoid these "distinctions without a difference"

→ Left-most derivations

→ Say $w A\alpha \underset{lm}{\Rightarrow} w\beta\alpha$ if w is a string of terminals only and $A \to \beta$ is a production

→ Also $\alpha \overset{*}{\Rightarrow}_{lm} \beta$ if α becomes β by a sequence of 0 or more $\Rightarrow_{lm}$ steps.

→ Eg: Balanced - Parentheses grammar :

$$S \rightarrow SS \mid (S) \mid ()$$

Let $w = (())()$

$S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)() \Rightarrow (())()$

$S \Rightarrow SS \Rightarrow S() \Rightarrow (S)() \Rightarrow (())()$

Leftmost derivation:

$S \Rightarrow SS \Rightarrow (S)S \Rightarrow (())S \Rightarrow (())()$

$S \overset{*}{\Rightarrow}_{lm} (())()$

Parse Trees :

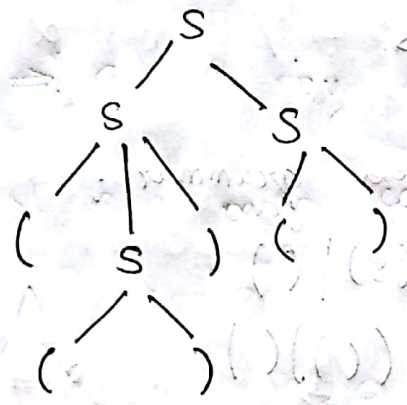
→ Parse trees are trees labeled by symbols of a particular CFG.

→ Leaves :
- labeled by terminals or ϵ

→ Interior Nodes :
- labeled by a variable
  - → Children are labeled by the right side of a production for the parent.

→ Root :
must be labeled by the start symbol

Eg:



yield = (()) ()

Yield of a Parse Tree

→ The concatenation of the labels of the leaves in left-to-right order

→ That is, in the order of a pre order traversal is called the yield of the parse tree

Parse Trees, Left and Rightmost Derivations :

→ For every parse tree, there is a unique leftmost and a unique rightmost derivation

→ We will prove :

① If there is a parse tree with root labeled A and yield w then $A \overset{*}{\underset{lm}{\Rightarrow}} w$

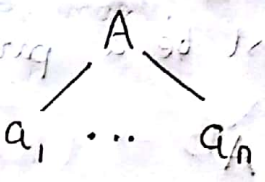
② If $A \overset{*}{\underset{lm}{\Rightarrow}} w$ then there is a parse tree with root A and yield w

PROOF FOR PART 1:

→ Induction on the height (length of the longest path from the root) of the tree

Basis:

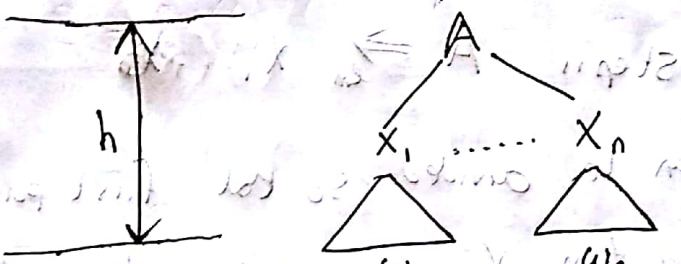
height 1 tree looks



- $A \to a_1 \dots a_n$ must be a production

Thus $A \overset{*}{\underset{lm}{\Rightarrow}} a_1 \dots a_n$

→ Assume (I) for trees of height < h, and let this tree have height h



By IH,

$X_i \overset{*}{\underset{lm}{\Rightarrow}} w_i$

- Note: if X_i is a terminal, then $X_i = w_i$

→ Thus,

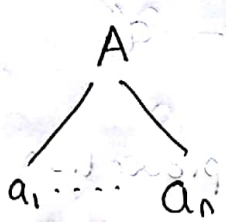
$$A \underset{lm}{\Rightarrow} X_1 \dots X_n \overset{*}{\underset{lm}{\Rightarrow}} w_1 X_2 \dots X_n$$

$$\overset{*}{\underset{lm}{\Rightarrow}} w_1 w_2 X_3 \dots X_n \overset{*}{\underset{lm}{\Rightarrow}} \dots$$

$$\overset{*}{\underset{lm}{\Rightarrow}} w_1 w_2 \dots w_n$$

Proof for Part 2 :

→ If $A \Rightarrow_{lm}^{*} a_1 \ldots a_n$ by a one-step derivation then there must be a parse tree.



→ Assume Part 2 to be true for derivations of fewer than $k > 1$ steps, and let $A \Rightarrow_{lm}^{*} w$ be a k-step derivation

→ First step is $A \Rightarrow_{lm} X_1 \ldots X_n$

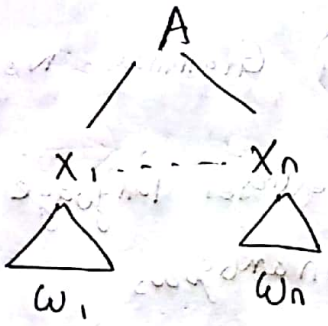
(w can be divided so that first portion is derived from X_1, the next is derived from X_2, and so on)

If X_i is a terminal, then $w_i = X_i$

→ That is, $X_i \Rightarrow_{lm}^{*} w_i$ for all i such that X_i is a variable and the derivation takes fewer than k-steps.

→ By the IH, if X_i is a variable then there is a parse tree with root X_i and yield w_i

→ Thus there is a parse tree



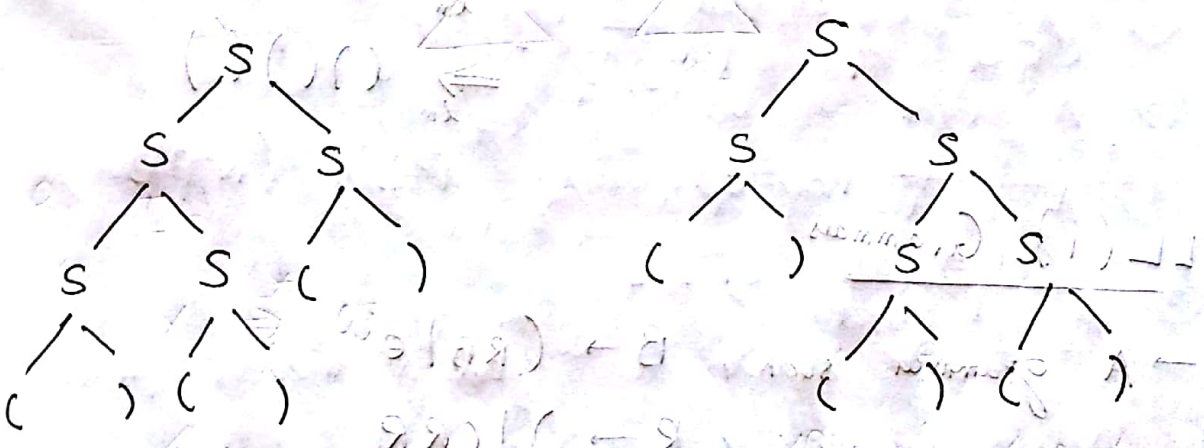
AMBIGIOUS GRAMMARS : 1/11/18

→ A CFG is ambiguous if there is a string in the language that is the yield of two or more parse trees

Eg:

$$S \rightarrow SS \mid (S) \mid ()$$

Two parse trees for ()()() are



An Equivalent Definition of ambiguous CFGs is

- There is a string in the language that has two different leftmost derivations
- There is a string in the language that has two different rightmost derivations.

→ Ambiguity is a Property of Grammars, Not Languages

→ For balanced parentheses language, here is an CFG, which is unambiguous

The start symbol deriving balanced strings

$B \to (RB \mid \epsilon$

$R \to) \mid (RR$

generates strings that have one more right parentheses than left

Consider $w = ()()()$

$B \Rightarrow_{lm} (RB \Rightarrow_{lm} ()B$

$\Rightarrow_{lm} ()(RB \Rightarrow_{lm} ()()B$

$\Rightarrow_{lm} ()()(RB$

$\Rightarrow_{lm} ()()()B$

$\Rightarrow_{lm} ()()()$

This process is called Parsing

LL(1) Grammars.

→ A grammar such $B \to (RB \mid \epsilon$

$R \to) \mid (RR$

where you can always figure out the production to use in a leftmost derivation by scanning the given string left-to-right & looking only at the next one symbol, is called

→ "leftmost derivation, left-to-right scan, one symbol of lookahead"

(L) (L) (1) LL(1)

→ Most programming languages have LL(1) grammar

→ LL(1) grammars are never ambiguous.

Inherent Ambiguity

→ Certain CFL's are inherently ambiguous, meaning that every grammar for the language is ambiguous

Eg:

$$\{0^i 1^j 2^k \mid i=j \text{ or } j=k\}$$

$S \rightarrow AB \mid CD$ (S: start symbol)

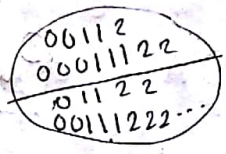
$A \rightarrow 0A1 \mid 01$

$B \rightarrow 2B \mid 2$

$C \rightarrow 0C \mid 0$

$D \rightarrow 1D2 \mid 12$

Eg:



Eg: $\omega = 012$ (here $i=j=k$)

$$S \underset{lm}{\Rightarrow} AB \Rightarrow 01B \underset{lm}{\Rightarrow} 012$$

$$S \underset{lm}{\Rightarrow} CD \Rightarrow 0D \underset{lm}{\Rightarrow} 012$$

Chomsky Normal Form (CNF)

→ A CFG is said to be in CNF if every production is of one of these forms:

① $A \rightarrow BC$ (right side is of 2 variables)

② $A \rightarrow a$ (right side is of single variable)

Theorem :

If L is a CFL then $L-\{\epsilon\}$ has a CFG in CNF

To read from book.

CYK Parsing Algorithm

$O(n^3 \cdot |G|)$

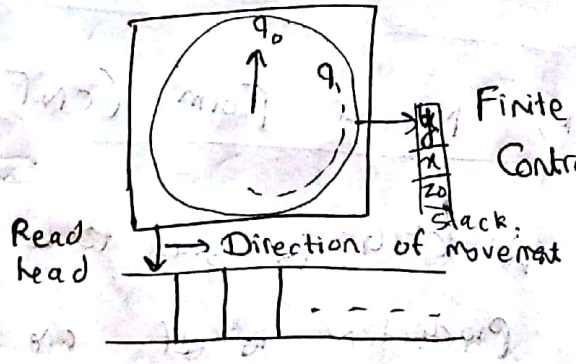
Push Down Automata (PDA)

→ The PDA is an automaton equivalent to the CFG in language-defining power.

→ Only the non-deterministic PDA (NPDA) defines all CFL's

→ But the deterministic version models parsers

→ Most programming languages have deterministic PDA's



$$\{ ww^R \mid w \in \{0,1\}^* \}$$

Intuition : PDA

→ Think of an ϵ-NFA with additional power that it can manupulate a stack

→ Its moves are determined by

① The current state (of its "NFA")

② The current i/p symbol (or ϵ) and

③ The current symbol on top of its stack

→ Being non-deterministic, the PDA can have a choice of next moves

→ In each choice, the PDA can:

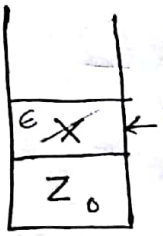
① Change state, and also

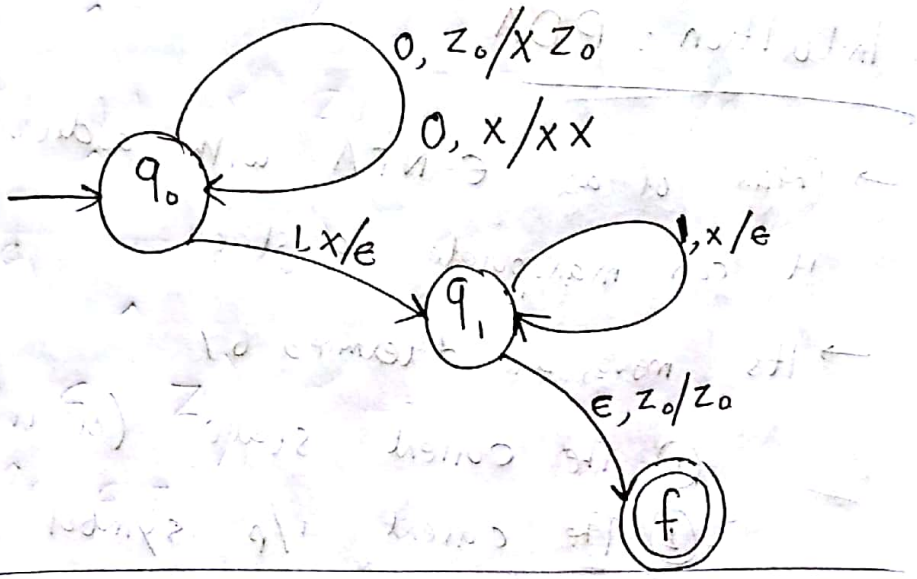
② Replace the top symbol on the stack by a sequence of zero or more symbols

✓ "zero symbols" = "pop"

✓ Many symbols = sequence of pushes

Eg: A PDA for accepting $\{0^n 1^n \mid n \geq 1\}$





A PDA is described by : 05/11/18

1. A finite set of states (Q, typically)
2. An input alphabet (Σ, typically)
3. A stack alphabet (Γ, typically)
4. A transition function (δ, typically)
5. A start state (q_0, in Q, typically)
6. A start symbol (Z_0 in Γ, typically)
7. A set of final states ($F \subseteq Q$ typically)

The Transition Function

→ takes three arguments

1. A state in Q
2. An input, which is either a symbol in Σ or ϵ

3. A stack symbol in Γ

→ $\delta(q, a, z)$

= a set of zero or more actions of the form

(p, α)

p → a state, α → a string of stack elements.

Instantaneous Descriptions (IDs)

→ An ID is a triple (q, w, α) where:

1. q is the current state
2. w is the remaining input
3. α is the stak contents, top at the left.

→ The "Goes-To" Relation

→ To say that ID I can become ID J in one move of the PDA, we write

$I \vdash J$

→ Formally, $(q, aw, X\alpha) \vdash (p, w, \beta\alpha)$ for any w and α if $\delta(q, a, X)$ contains (p, β)

Eg: $w = 0011$

$(q_0, 0011, Z_0)$ (from state diag)

$\vdash (q_0, 011, XZ_0)$

$\vdash (q_0, 11, XXZ_0)$

$\vdash (q_1, 1, XZ_0)$

$\vdash (q_1, \epsilon, Z_0)$ → m/c has read the string & reached final state of m/c

$\vdash (q_f, \epsilon, Z_0)$ (No action over the stack top)

So Accept.

Eg: $w = 001$

$(q_0, 001, Z_0)$

$\vdash (q_0, 01, XZ_0)$

$\vdash (q_0, 1, XXZ_0)$

$\vdash (q_1, \epsilon, XZ_0)$ Reject

$w = 011$

$(q_0, 011, Z_0)$

$\vdash (q_0, 11, XZ_0)$

$\vdash (q_0, 1, Z_0)$

Reject (∵ No move defined)

So rejection can happen after reading whole i/p or after halting at a non-final state)

Language of a PDA :

→ The common way to define the language of a PDA is by FINAL STATE.

→ If P is a PDA, then $L(P)$ is the set of strings w such that

$(q_0, w, Z_0) \vdash^* (f, \epsilon, \alpha)$ for final state f and any α.

⇒ Another way to define the language accepted by the same PDA is by empty stack.

→ If P is a PDA, then $N(P)$ is the set of strings w such that

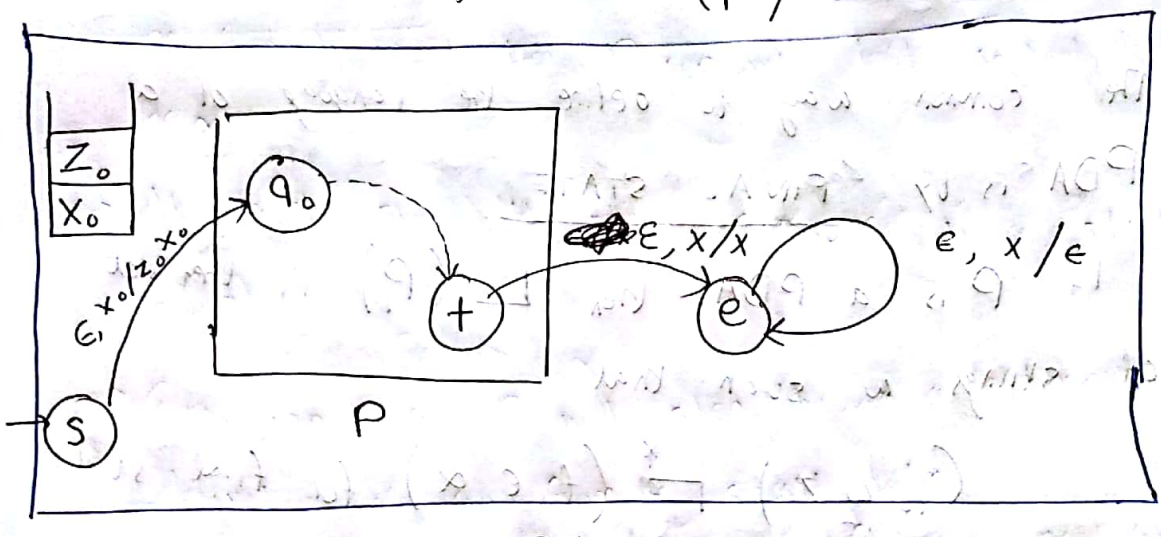
$(q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon)$ for any state q.

Equivalence of Language Definitions

(1) If $L = L(P)$, then there is another PDA P' such that $L = N(P')$

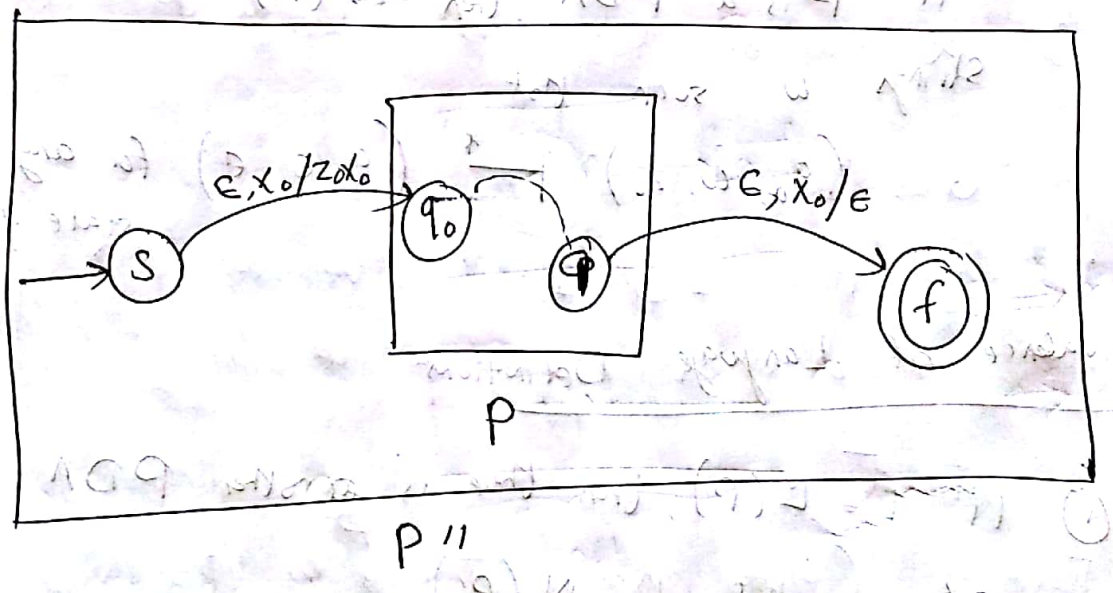
(2) If $L = N(P)$, then there is another PDA P'', such that $L = L(P'')$

PROOF: $L(P) \longrightarrow N(P')$



$N(P') = L(P)$

PROOF: $N(P) \longrightarrow L(P'')$ There is a PDA P that accepts some lang by empty stack. There is a same lang L s.t.



S_0, ~~$L(P) \quad N(P)$~~ $N(P) = L(P'')$ (from the 2 proofs)

Q Design a PDA to accept

$\{\omega\omega^R \mid \omega \in \{0,1\}^+\}$ (ω^R = reverse)

ω = any binary string

Deterministic PDA

→ To be deterministic, there must be at most one choice of move for any state q, input symbol a, and stack symbol X.

→ In addition, there must not be a choice between using input ϵ or real input.

→ Formally, $\delta(q, a, X)$ and $\delta(q, \epsilon, X)$ cannot both be non-empty.

14/11/18

Equivalence of PDA, CFG

→ Converting a CFG to a PDA

→ Let $L = L(G)$

→ Let us construct a PDA P such that

$N(P) = L$

∴ P has:

- One state q
- Input symbols = terminals of G
- Stack symbols = all symbols of G
- Start symbol = start symbol of G

Intuition About P

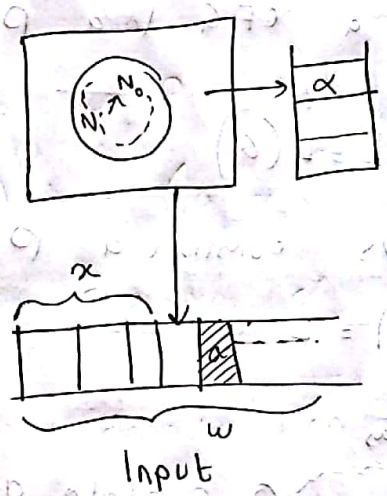
→ At each step, P represents some sentential form (step of a left most derivation)

→ If the stack of P is α, and P has consumed so far x from its input, then P represents left sentential form $x\alpha$.

→ At empty stack, the input consumed is a string in $L(G)$

→ Given input ω, P will step through a leftmost derivation of ω from the start state.

→ Since P cannot know, what this derivation is, or even what the end of ω is, it uses non-determinism to "guess" the production to use at each step.



$S \overset{*}{\underset{lm}{\Rightarrow}} x\alpha$

$\overset{*}{\underset{lm}{\Rightarrow}} xa\beta$ (with one or more steps)

Transition Function of P,

① If $\delta(q, a, a) = (q, \epsilon)$ (Type 1 rule)

- This step does not change the left sentential form represented, but "moves" responsibility for a from the stack to the consumed input

(2) If $A \to \alpha$ is a production of G, then

$\delta(q, \epsilon, A)$ contains (q, α) ... (Type 2 rule)

- Guess a production for A, and represent the next left sentential form in the derivation.

$$A \to \gamma$$

$$S \overset{*}{\underset{lm}{\Rightarrow}} x A \beta \quad (\alpha)$$

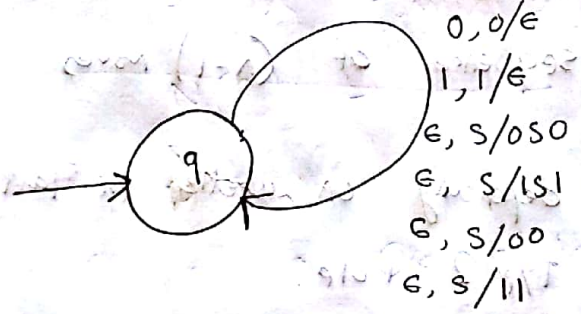
$$\Rightarrow x \gamma \beta$$

Eg. Give a PDA for grammar G, given as:

(if it starts with 0, it ends with 0; if it starts with 1, it ends with 1)

$$S \to 0S0 \mid 1S1 \mid 00 \mid 11$$

$L(G) = \{ ww^R \mid w \text{ is in } \{0,1\}^+ \}$ — even length binary palindromes.



Eg: 0011
0110

~~$(q, 0011, SZ_0) \vdash (q, 0011)$~~

~~(0 0)~~

$(q, 0110, SZ_0) \vdash (q, 0110, 0S0Z_0)$

$\vdash (q, 110, S0Z_0) \vdash (q, 110, 10Z_0)$

$\vdash (q, 10, 10Z_0) \vdash (q, 0, 0Z_0) \vdash (q, \epsilon, Z_0) \Rightarrow$ Accepts

PDA has read the entire ip string and by that time it has made its stack empty

Proof $L(P) = L(G)$ — 15/11/18

We need to show that

$(q, wx, S) \vdash^* (q, x, \alpha)$ for any x if and only if $S \underset{lm}{\Rightarrow}^* w\alpha$

Part 1:

If $(q, wx, S) \vdash^* (q, x, \alpha)$ for any x,

then $S \underset{lm}{\Rightarrow}^* w\alpha$

Every triplet is called Instantaneous Description (ID) of PDA

Basis: 0 steps (i.e. no moves).

→ Then $\alpha = S$, $w = \epsilon$ and $S \underset{lm}{\Rightarrow}^* S$ is true

→ Consider n moves of P:

$(q, wx, S) \vdash^* (q, x, \alpha)$ and assume the Inductive Hypothesis for sequences of $(n-1)$ moves

→ There are 2 cases depending on whether the last move uses a Type 1 or Type 2 rule.

→ Use of Type 1 rule:

→ The move sequence must be of the form:

$(q, \underbrace{ya}_{w} x, S) \vdash^* (q, ax, a\alpha)$

$\vdash (q, x, \alpha)$ where $ya = w$

By the IH applied to first $(n-1)$ steps, | where $ya = \omega$

$$S \overset{*}{\underset{lm}{\Rightarrow}} ya\alpha$$

But $ya = \omega$, So $S \overset{*}{\underset{lm}{\Rightarrow}} \omega\alpha$

→ Use of Type 2 rule

→ The move sequence must be of the form

$$(q, \omega x, s) \overset{*}{\vdash} (q, x, A\beta)$$

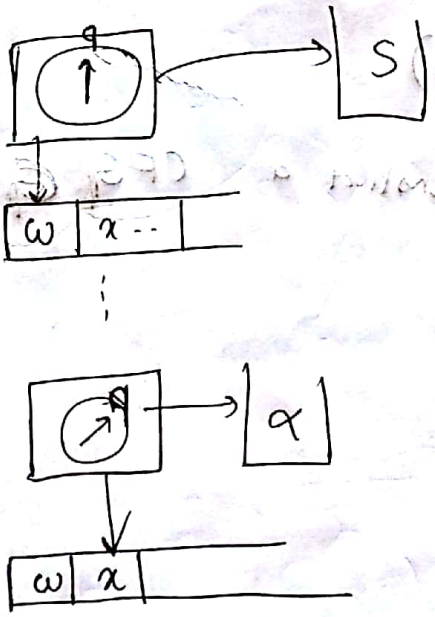
$$\vdash (q, x, \gamma\beta)$$ where

$A \to \gamma$ is a production and $\alpha = \gamma\beta$

→ By the IH applied to first $(n-1)$ steps,

$$S \overset{*}{\underset{lm}{\Rightarrow}} \omega A\beta$$

Thus $S \overset{*}{\underset{lm}{\Rightarrow}} \omega\gamma\beta = \omega\alpha$



$$S \overset{*}{\underset{lm}{\Rightarrow}} \omega\alpha$$

Proof of part 2:

If $S \overset{*}{\underset{lm}{\Rightarrow}} \omega\alpha$, then $(q, \omega x, S) \overset{*}{\vdash} (q, x, \alpha)$

~~Proof of Part 2~~ : Proof - Completion

~~If $\vdash^*$~~

→ We now have,

$(q, wx, S) \vdash^* (q, x, \alpha)$

if and only if $S \underset{lm}{\Rightarrow^*} w\alpha$

→ In particular, let $x = \alpha = \epsilon$,

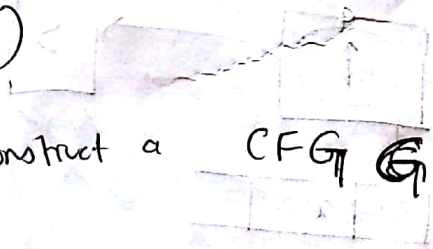
Then $(q, w, S) \vdash^* (q, \epsilon, \epsilon)$ if and only if $S \underset{lm}{\Rightarrow^*} w$

that is w is in $N(P)$ if and only if w is in $L(G)$

From PDA to a CFG

→ Assume $L = N(P)$

we will construct a CFG G such that $L = L(G)$



Pumping Lemma for CFLs :

16/11/18

→ For every CFL L, there is an integer n, such that for every string z in L of length ≥ n, there exists $z = uvwxy$ such that :

1. $|vwx| \leq n$
2. $|vx| > 0$
3. for all $i \geq 0$, uv^iwx^iy is in L

PROOF:

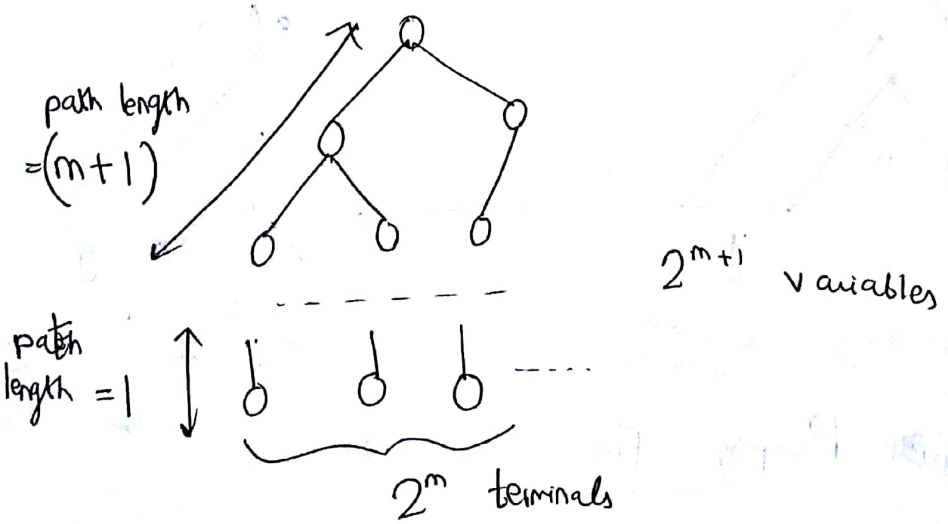
→ Start with a CNF grammar for $L - \{\epsilon\}$

→ Let the grammar have m variables

→ Pick $n = 2^m$

→ Let $|z| \geq n$

→ We claim that a parse tree with yield z must have a path of length m + 2 or more

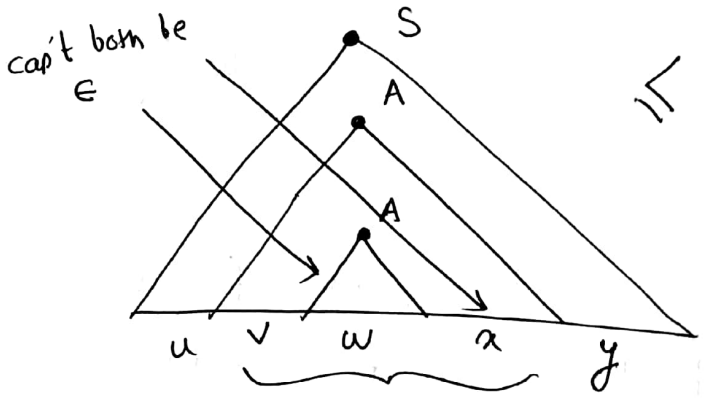


→ Now we know that the parse tree for z has a path with at least $(m+1)$ variables

→ Consider some longest path

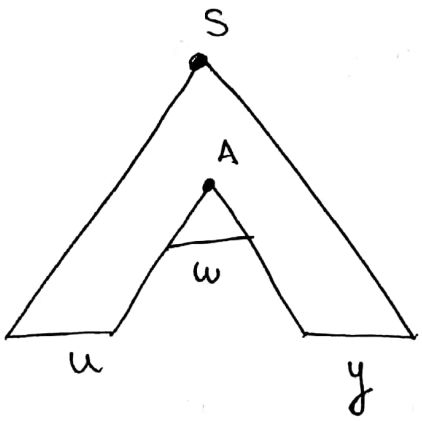
→ There are only m different variables, so among the lowest $(m+1)$ we can find two nodes with the same label, say A.

→ The parse tree thus looks like:



$\leq 2^m = n$ because a longest path chosen

Pump zero times

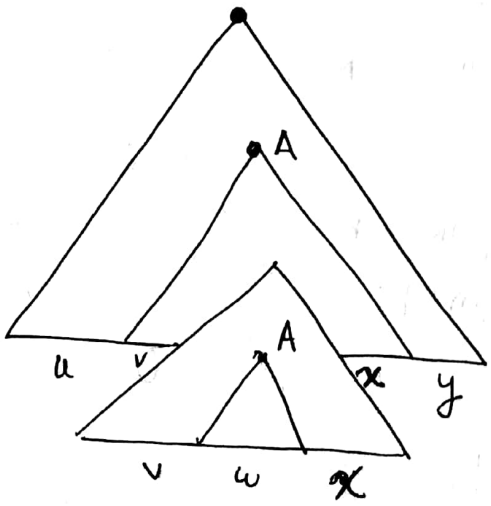


yield = uwy

of the subtree

This belongs to the lang.

Pump Twice



yield: uv^2wx^2y

Again This belongs to the langu

Pumping Thrice etc. etc. . . .

uv^3wx^3y - - - -

Using pumping lemma, prove that $\{ww \mid w \text{ is in } \{0,1\}^*\}$ is not a CFL

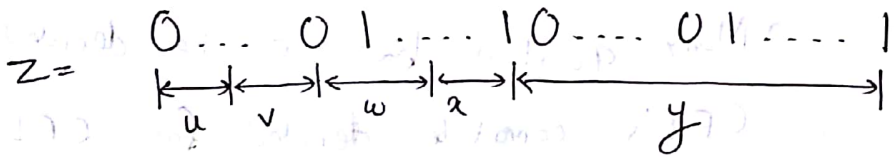
w is any binary string including the empty string

Let us initially assume that $\{ww \mid w \text{ in } \{0,1\}^*\}$ is a CFL.

Let m be the constant of Pumping Lemma

and $z = 0^m 1^m 0^m 1^m$ (length of string > constant of the Pumping Lemma)

vwx can be chosen in many ways



For the ~~choice~~ choice above we can get string of the form

for $i = 0$,

$0^k 1^j 0^m 1^m$ where $k < m$ or $j < m$ which is not in L

In this way, we can show that this lang. is not a CFL

22/11/18

Properties of CFL's:

→ Decision Properties

→ There are algorithms to decide if

(*) (1) string w is in CFL L

(2) CFL is empty.

(3) CFL L is infinite

→ Non-Decision / Undecidable Properties

→ Many question that can be decided for CFL's cannot be decided for CFL's

Eg:

- Are two CFL's the same?
- Are two CFL's disjoint?

Need theory of Turing machines & decidability to prove no algo exists

(*) For (1),

CNF for CFG's

$A \to a$

$A \to CD$

$A, C, D \in V$

$a \in T$

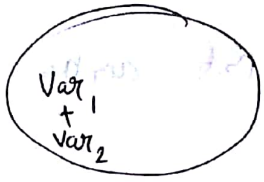
When the grammar is in CNF, we can apply CYK Parsing Algo (notes as pic) which runs in time $O(n^3)$ where $n = |w|$ and checks whether string w belongs

for ② CFL L is empty

→ Identify the variables which do not derive any grama

$Var_1 \longrightarrow t^+$

$Var_2 \longrightarrow t^+ var_1 \mid var_1 \mid var_1 t^+$



Variables which derive terminal strings

The ~~strings~~ var thus left are the ones which do not derive terminal strings

→ Derive the var which does not derive terminal strings

→ If the stat symbol is eliminated, then L is empty

For ③ CFL L is infinite

read notes (as given pic)

→ Closure Properties of CFL's under

→ Union

→ Concatenation

→ Star

→ Reversal

→ Homomorphism

But CFL are not closed under the operation of intersection.

Union

CFL's

L_1 & L_2

$L_1 \cup L_2 =$ a CFL

set of terminal ← (Σ_1, V_1) → set of variables

Let for L_1, there is a CFG G_1 ; start symbol = S_1

L_2 - - - - - G_2 ; start symbol = S_2
(Σ_2, V_2)

We show that another CFG G_3 can be formed.

So, we hv. constructed another CFG.

$\Sigma_1 \cup \Sigma_2$

set of variables: $V_1 \cup V_2 \cup \{S\}$

Set of productions: $P = \{S \to S_1 \mid S_2\} \cup P_1 \cup P_2$

$S \Rightarrow_{lm} S_1 \overset{*}{\Rightarrow}_{lm}$ a string of L_1

$S \Rightarrow_{lm} S_2 \overset{*}{\Rightarrow}_{lm}$ a string of L_2

So from S_1 a string of L_1 can be derived, a string of L_2 can be

So CFL, L_1 and L_2 are closed under union

start state of G_3.

Concatenation : $S \to S_1 S_2$ (leading part consisting of strings of L_1, trailing part consisting of strings of L_2)

Hence CFGs are closed under the operation of concatenation

Star

$S \rightarrow S_1 S \mid \epsilon$

additional start symbol to the string of grammar G_1,

finally we derive a string of strings of G_1,

Hence CFL are closed under star op. of closure

Reversal : Reverses RHS of all production rules of that grammar.

All strings derived will be the reverse of the string $\in L$,

So CFL are closed under Reversal

Homomorphism