INTRODUCTION

- Automata
- What does it mean to suppose a language

AUTOMATA THEORY

Automata is the plural of automaton i.e. a machine

A computer's abstract definition : Different spaces/category/class of computers that can compute/helpful for a class of problem

Two characteristics of machines

Intrinsic : Structure, how it is made → what it can do, its comp

Extrinsic :

Formal languages

Power of computer comes from the no. of languages it can deal with / recognize a language

Def. of Machine

Computation is transformation of an i/p to an o/p

Review

$\aleph_0$ and $\aleph_1$ have 1-1 mapping

$\aleph_0$ is natural nos

- Elementary set theory
  - $\mathbb{N} = \{0, 1, 2 \ldots\}$
  - enumeration (size) & comprehension
  - $P(A) = \{B \mid B \subseteq A\}$; the power set

There are two types of infinity) Countable (denumerable) $\aleph_0$ & Uncountable (Real nos.) $P(\aleph_0)$

- You can't solve all problems : Halting problem : no computer can be know whether a program stops/not : Alan Turing
  (whether a statement is T/F)
- Computability theory

LANGUAGE

Eg:

Σ - Alphabet Σ : A finite collection of symbols 0/1 — $\Sigma = \{a, b\}$

ε Word/String on Σ: Any finite sequence of alphabetic symbols from Σ — $\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, \ldots\}$ (Countable)

Σ^* $\Sigma^* = \{w \mid w \text{ is a word over } \Sigma\}$ Language

Identity/Monoid: Algebraic structure that has a set, associative closed operation (and concatenation) & an identity element (Empty string)

Ref: p3

Computers mean finite subsets of Σ^*

INDUCTIVE SETS

Ch 0 of book

Feynman

MACHINE : Mathematic definition

$S = \langle X \; X^\circ \; U \; \rightarrow \rangle$ → everything is fixed . moves from one state to another

where X : State space Set of all possible states . reacts to i/p

X° : $x^\circ \subseteq X$ Initial space . always in a given state x

Turing Machine

U : Input space

$\rightarrow$: $\rightarrow \subseteq X \cdot U \cdot X$ Relation

For same i/p, state can go to diff state Model is a useful approx

transition function is a function

Rules of the tournament How actual i/p is unknown

DETERMINISTIC FINITE STATE AUTOMATON

Built a model with features relevant/change State variable

$M = \langle Q, q_0, \Sigma, \delta: Q \times \Sigma \rightarrow Q, F \subseteq Q \rangle$ δ cannot change Σ

A machine is a collection of states Q : Set of states

It moves from one state to another, we assume that the state space is finite Multiple q_0 does not change anything

q_0 is the state where the machine is at the beginning $q_0 \in Q$, i.e. initial state

Σ : finite set of symbols / the action that causes the change of state / input alphabet

$\delta: Q \times \Sigma \rightarrow Q$ δ : the rule that determines which state to go from which state according to current state Q & impulse Σ / transition function

F : final state, usually ≥ 1 / terminal state Set Arbitrary : we just say that F is fixed

We like it to end at final state

→

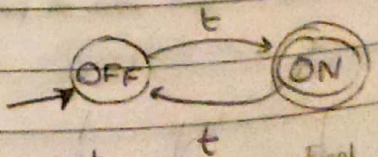
Eg: Laptop as a machine LAPTOP is a FINITE STATE MACHINE

State Space : All possible (stat) memory configurations

Input : Interrupts of i/p devices, networks etc. can sit in the memory

δ : Set of instructions (in machine code) like reg write, send packets etc.

Eg2: Switch . An ideal switch



$Q = \{ON, OFF\}$

q_0 = OFF $F = \{ON\}$ Final

Σ = toggle $\{t\}$ inputs

Dynamics of system $\delta(ON, t) = OFF$ $\delta(OFF, t) = ON$

Input String $w \in \Sigma^*$ $\Sigma^* = \{\varepsilon, t, tt, ttt \ldots$ Each element is finite, but set is infinite

This is a dead machine . Now let's run it : Behaviour of a machine wrt input string

Eg: if $w = ttt$, state is OFF $\xrightarrow{t}$ ON $\xrightarrow{t}$ OFF $\xrightarrow{t}$ ON

Final state

A

2

q

no. of inputs

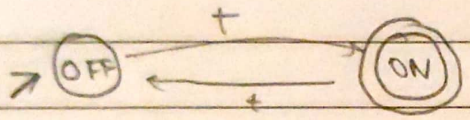
With a model we can predict the behaviour of machine on an input string

If An ∞ set, is it necessary to have an element of infinite length
→ It has 1→1 correspondance to N since ∞ ∉ N.

INPUT STRING or INPUT ALPHABET

Input string : String with character $w \in \Sigma^*$

FINITE STATE DIAGRAM of the SWITCH



If all i/ps are obtained beforehand, then i/p string is finite

* i/p can be provided during run-time but in this case we don't know whether i/p is done/not ⇒ machine runs ∞
Apart that, it is same

Dynamics

i	Input buffer (Current Input)	State (current state)
0	tt	OFF
1	t	ON
2	ε	OFF

Predictive ability

* ~~A machine accepts a string, only when the~~

ACCEPTANCE OF A STRING BY A MACHINE

A machine accepts an input string if after running that string on the machine, the end result is a final state.

~~Driver~~ DRIVER : Generalised Automaton

The automaton program that takes in an automaton & input, runs the automaton on the input, to produce an o/p

Mathematical def of acceptance of a string by a DFA

A DFA $A = \langle Q, q_0, \Sigma, \delta, F \rangle$, $q_0 \in Q$, $F \subseteq Q$, $\delta : Q \times \Sigma \to Q$ accepts a string $\omega \in \Sigma^*$ if there is a sequence $q_0 w_0 q_1 w_1 \ldots q_{n-1} w_{n-1} q_n$ such that (trace)

0) $q_0 = q_I$

1) $w_0 w_1 \ldots w_{n-1} = \omega$

2) $\delta(q_i, w_i) = q_{i+1}$ for $0 \le i < n$ Should respect the dynamics of the system

3) $q_n \in F$ there are transitions defined on q_n

Not a universal driver

DRIVER As an Automaton : Run only DFA

Motivation : take out the human of the process i.e. 1) automate the running 2) to build universal machines

Q. Define an automaton that simulates the running the machine on some input - i.e. Driver

Eg= makefile, OS, browser, compiler

Consider a DFA $A = \langle Q\ q_I, \Sigma, \delta, F \rangle$ & input w

— A driver automaton that is parameterised over a DFA A and an input string w

~~Q the driver~~ ; ~~Q × Σ~~ Arbitrary machine $D = \langle X_D, X_D^\circ, U_D, \xrightarrow{D} \rangle$

current state, current i/p str

$X_D = Q \times \Sigma^*$ $\quad x_D = (q, w) \in X_D$

$X_D^\circ = \{q_I\} \times \Sigma^*$ Initialize the automata with q_I

$U_D = \{next\}$ Clock ticking It is autonomous $U_D = \langle Tick \rangle$

$\xrightarrow{D}$ =

$x_D \xrightarrow{tick} x'_D$

$(q, \varepsilon) \xrightarrow{tick}$

$(q, aw) \xrightarrow{tick} (\delta(q,a), w)$

$w \in \Sigma^* \quad a \in \Sigma$

$(q, \varepsilon) \xrightarrow{tick} \not\rightarrow$ clock ticks

[Signals the end of run, Driver stops]

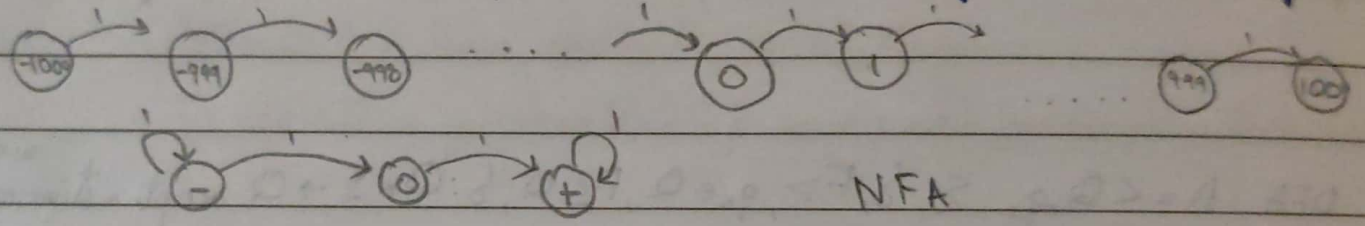
[consumes one letter a, moves to next state, then reduces i/p str by 1 letter]

NON - DETERMINISM -

Consider a DFA with states, -1000, -999 ... -1, 0, 1, ... 999, 1000 and a transition function (add 1)

Suppose we do not have enough storage to store 2000 states and so we use 3 states, positive, negative and

∴ Adding 1 at negative state has 2 possibilities: either 0 or still negative : Abstract Analysis



REACHABILITY

Let $S = \langle X, U, \rightarrow \rangle$ be an arbitrary machine where $\rightarrow \subseteq X \cdot U \cdot X$ i.e. $(x, u, x') \leftrightarrow : x \xrightarrow{u} x'$

x' is reachable from x if $\exists\ x_1 \dots x_{n-1} \in X$ such that $x \xrightarrow{u} x_1$, $x_1 \xrightarrow{u_2} x_2$, ..., $x_{n-1} \xrightarrow{u_n} x_n$ i.e $x \xrightarrow{u_1 \dots u_n} x_n$ labelled directed graph

ACCEPTANCE OF A STRING BY A DFA

Let $M = (Q, \Sigma, q_0, \delta, F)$ be a DFA. Let $w \in \Sigma^*$

M accepts w if $\exists\ q \in F$ $q_0 \xrightarrow{w} q$

$q \xrightarrow{a} q'$ if $q' = \delta(q, a)$

Acceptance is important only for a language

Non-deterministic Finite State Machine

$N = (Q, q_0, \Sigma, \delta, F)$ is said to be a NFA where

$q_0 \in Q$

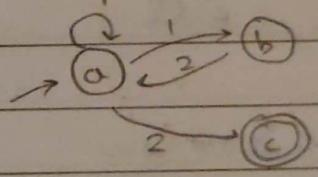
δ : $\delta : Q \times \Sigma \to 2^Q$ $\quad 2^Q$ is the notation for power set

$\delta \subseteq Q \times \Sigma \times Q$

Example

Let $Q = \{a, b, c\}$ and $\Sigma = \{1, 2\}$ and $\delta = \{(a,1,a), (a,2,c), (b,2,a), (a,1,b)\}$

	a	b	c
1	{a,b}	{ }	{ }
2	{c}	{a}	{ }



Exceptions are handling is machine specific: No state transition can take place / move to error state in case of invalid i/p. i.e this model is inadequate for modelling the situation when 1 or 2 i/p is given at c.

ACCEPTANCE OF A STRING BY AN NFA

Let $N = (Q, q_0, \Sigma, \delta, F)$ be an NFA. Let $w \in \Sigma^*$

N accepts w iff $\exists\ q \in F$ such that $q_0 \xrightarrow{w} q$ i.e. q is reachable from q_0

[String accepted]

LANGUAGE RECOGNISED BY AUTOMATON

Language recognized by an automata is the set of all input strings accepted by the machine / automaton.

THEOREM

1) ~~A~~ Given a ~~DFA~~ NFA that recognizes a language L, show that ∃ a DFA that recognizes the same language
2) Vice Versa of part 1
3) DFA is equivalent to NFA

Example.

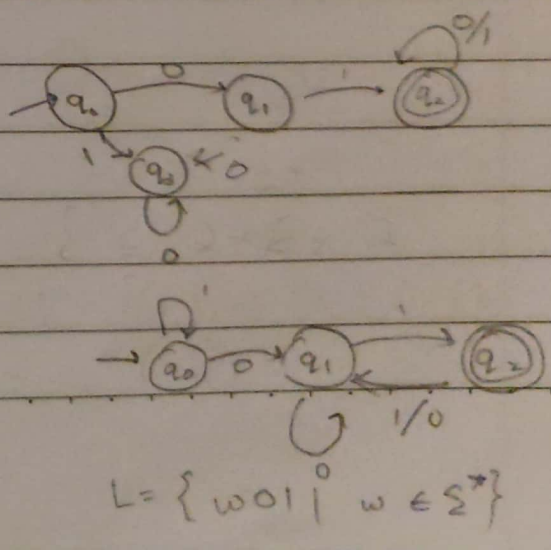
a) Consider a DFA with $\Sigma = \{0, 1\}$

~~Ad~~ Accepted strings are 01, 0101, 01011 ....

i.e. Language $L(A) = \{01\ w \mid w \in \Sigma^*\}$

Characterisation of DFA is enumerating the accepted string

i/p	0	ε	1	10
	N/A	N/A	N/A	N/A



NFA is more intuitive

THEOREM Language Equivalence

$\forall$ NFA N, $\exists$ DFA M such that $L(N) = L(M)$

Proof

Let $N = (Q, q_0, \Sigma, \delta, F)$ be an NFA [$q \in \delta(q, a)$ is denoted by $q \xrightarrow{a} q'$]

N accepts w if $\exists\ q_1, q_2 \dots q_n$ such that 1) $q_1 \xrightarrow{w} q_2 \dots \xrightarrow{w} q_n$ 2) $q_n \in F$

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

a) $Q' = 2^Q$ [Power set of Q]

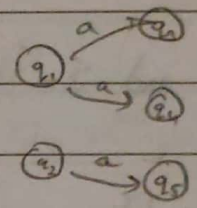
b) $q_0' = \{q_0\} \in Q'$

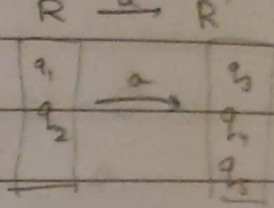
c) $\Sigma' = \Sigma$

d) $\delta' : Q' \times \Sigma \to Q'$

Consider $R \in Q'$, $\delta'(R, a) \overset{\text{defined}}{=} \bigcup_{r \in R} \delta(r, a)$

i.e. $R \xrightarrow{a} R' \Rightarrow \forall\ r' \in R'\ \exists\ r \in R : r \xrightarrow{a} r'$



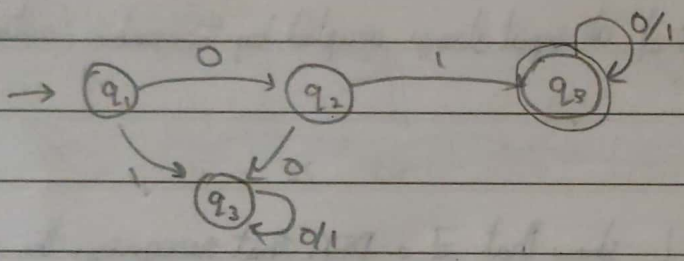


e) $F' = \{R \in Q \mid \exists q \in R \cap F\}$ [Sets containing a final state of the orig. machine]

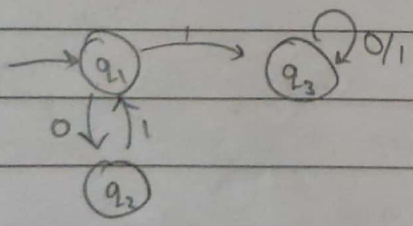
Verificate $\longrightarrow$

TUTORIAL

1. Design a DFA that accepts all input strings starting with 01 followed by $(01)(0+1)^*$ [any no. of 0s or 1s]

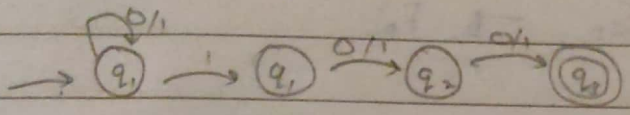


2. Design an NFA that accepts all strings starting with any no. of "01"s followed by a 1 and then any no. of 0s or 1s $(01)^*1(0+1)^*$



NFA & DFA have same power but diff rep

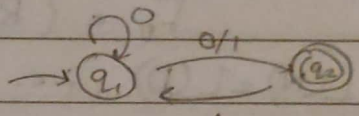
3. Design an NFA N with A be the language of N, of all strings over {0,1} containing 1 in the third position from the end. $A = (0+1)^* 1 (0+1)(0+1)$

→ 

Here there are 2 branches of computation. If in any branch NFA accepts the string, the string is accepted

Converting an NFA to DFA

1) If NFA has n states, DFA has 2^n states

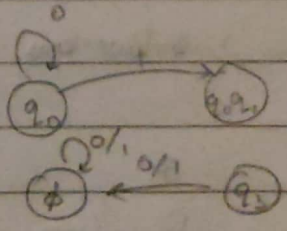
Eg: 2) Consider the NFA 

States of DFA are ϕ, q_1, q_0, q_0q_1

2)

	0	1
ϕ	ϕ	ϕ
q_0	q_2	$\{q_0, q_1\}$
q_1	q_2	q_2
q_2	q_3	q_3
q_3	ϕ	ϕ
$\{q_0q_1\}$	$\{q_0q_2\}$	$\{q_0q_1q_2\}$
$\{q_0q_3\}$	q_2	$\{q_0q_1q_3\}$
⋮		
$\{q_0q_1q_2\}$		
⋮		
$\{q_0q_1q_2q_3\}$		

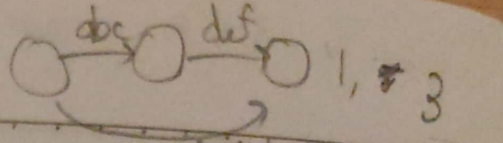
3) Remove all states for which there is o/p but no i/p



Start state NFA q_0 DFA q_0

Accept State NFA q_0 DFA q_0

NFA → DFA →



Verification (← Pg proof contd.

Given: Let NFA be $N = (Q_N, i_N, \Sigma_N, \rightarrow_N, F_N)$ $\delta_N(q,a) = \{\ldots\}$

Construct DFA $D = (Q_D, i_D, \Sigma_D, \rightarrow_D, F_D)$

where $Q_D = 2^{Q_N}$

$i_D = \{i_N\}$

$\Sigma_D = \Sigma_N = \Sigma$

(1), (3)
(1, 2) (2, 3) (1, 3)
(1, 2, 3)

$R, R' \in Q_D$, $R \rightarrow_D R'$ $R' = \bigcup_{r \in R} \delta_N(r,a)$

$F_D = \{ R \in Q_D \mid R \cap F \neq \phi \}$

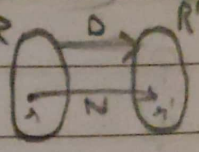
To show: $L(N) = L(D)$ i.e. $L(D) \subseteq L(N)$ and $L(N) \subseteq L(D)$

Summary

Trace (N): $i_N = r_0 \xrightarrow{a_1} r_1 \xrightarrow{a_2} r_2 \ldots \xrightarrow{a_N} r_N \in F_N$, then N accepts $w = a_1 a_2 \ldots a_N$

Trace (D): $i_D = \{i_N\} = R_0 \xrightarrow{a_1} R_1 \xrightarrow{a_2} R_2 ? \ldots ? \xrightarrow{a_N} R_N \in F_D$, $r_N \in R_N$ $r_N \in F_N$ D accepts w

$R \xrightarrow{a} R'$ // $R' = \bigcup_{r \in R} \{ r' \mid r \xrightarrow{a}_N r' \}$



→ LEMMA 1: If $r \in R$ & $r \xrightarrow{a}_N r'$, then $\exists$ R', 1) $r' \in R'$ and 2) $R \xrightarrow{a}_D R'$

? — PROOF: Take $R' = \bigcup_{r \in R} \{ r' : r \xrightarrow{a}_N r' \}$

1) By def of D, $R \xrightarrow{a}_D R'$

2) Suppose $r \xrightarrow{a}_N r'$, then $r' \in \delta(r,a)$, $r' \in R'$

$w \in \Sigma^*$ $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$

RULES: $r \xrightarrow{w}_N r'$ $w \in \Sigma^+$ $r, r' \in Q_N$ [Go in one/more steps] r can reach r' consuming w

$r \xrightarrow{a}_N r'$ ✗ ✗ ∈ rule $r \xrightarrow{a}_N r'$ | $r'' \xrightarrow{w}_N r'$ Transitive rule

RULES : $q \xrightarrow{w}_N q'$ $w \in \Sigma^+$ $q, q' \in Q_N$ [Go one/more steps] q can reach q' consuming w

$q \xrightarrow{a}_N q'$ $\Leftrightarrow$ Σ rule $\quad\quad$ $q \xrightarrow{a}_N q' \;|\; q'' \xrightarrow{w}_N q'$ Transitive rule.

$q \longrightarrow q'$ $\quad\quad$ $q \xrightarrow{aw}_N q'$

Accepted

LEMMA 1': Suppose $w \neq \varepsilon$ & $q \in R$ & $q \xrightarrow{w}_N q'$, then q can reach q'' consuming w.

Accepting run

then $\exists R'$ such that $R \xrightarrow{w}_D R'$ & $q' \in R'$

PROOF : ~~1. Base Case~~ : ~~$q \xrightarrow{w}_N q'$ via Σ~~ 1. $w \neq \varepsilon$ (Given)

~~2. $w = a$ for some~~ 2. $q \in R$ (Given)

B. 3. $q \xrightarrow{w}_N q'$

Base Case : 3. $q \xrightarrow{w}_N q'$ (Σ)

$w = a$ 4. $q \xrightarrow{a}_N q'$ for some $a \in \Sigma$

5. $\exists R' : R \xrightarrow{a}_D R'$ & $q' \in R'$ 4, 2, 1 using lemma 1

6. $R \xrightarrow{a}_D R'$ 5, Σ rule for D

Inductive Case : 1. $w = aw'$ for some $a \in \Sigma$, $w' \in \Sigma^+$

2. $q \xrightarrow{aw'}_N q'$ using TRANS

3. $q \xrightarrow{a}_N q''$ & $q'' \xrightarrow{w'}_N q'$ (Inversion

4. $\exists R''$ such that $q'' \in R''$ and $R \xrightarrow{a}_D R''$ (from L1)

5. $\exists R' : q' \in R'$ & $R'' \xrightarrow{w'}_D R'$ Using Induction Hypothesis, 3, 4

6. $R \xrightarrow{aw'}_D R'$ 4, 5 using Trans in D

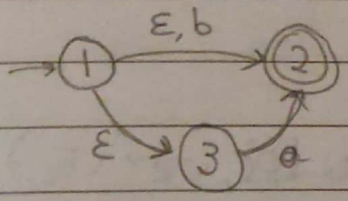
7. $R \xrightarrow{w}_D R'$ & $q' \in R'$

NFA WITH ε TRANSITIONS

Definition

An NFA/ε over an alphabet Σ: $\varepsilon \notin \Sigma$ is an NFA over $\Sigma \cup \{\varepsilon\}$

Example:



What does it mean to move from one state to another : $q \xrightarrow[E]{\varepsilon} q'$

[Graph] $q \xrightarrow[E]{\varepsilon} q'$ } Only 2 transitions are possible

$q \xrightarrow[E]{a} q'$, $a \in \Sigma$

Where can I go free (using ε transitions : $q \xrightarrow[E]{\varepsilon^* \; R^*} q'$ (Reflexive Transitive Relation) — Closure of

Reachable:

$R^* = \bigcup_{i \in N} R^i$

Consume an alphabet & do ε transitions before/after that $q \xrightarrow[E^*]{a} q'$

Suppose there is a fsm path b/w q_1 & q_2, $q_1 \xrightarrow[E]{\varepsilon^*} q_2$, $q_2 \xrightarrow[E]{a} q_3$, $q_3 \xrightarrow[E]{\varepsilon^*} q_4$

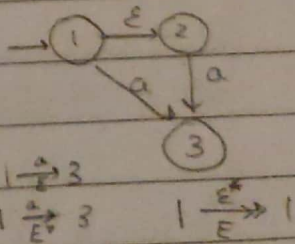
$q_1 \xrightarrow[E^*]{a} q_4$

Suppose i/p is $w \neq \varepsilon$, then $q \xrightarrow[E^*]{w} q'$ is the transitive closure of $\xrightarrow[E^*]{a}$

RULES : $\dfrac{q \xrightarrow[E^*]{a} q'}{q \xrightarrow[E^*]{a} q'}$ Σ $\qquad$ $\dfrac{q \xrightarrow[E^*]{a} q'' \quad q'' \xrightarrow[E^*]{w} q'}{q \xrightarrow[E^*]{aw} q'}$ TRANS

TRANSITIONS

1. $q \xrightarrow[E]{\varepsilon} q'$ | ε-transition in E
2. $q \xrightarrow[E]{a} q'$ non-ε-transition in E
3. $q \xrightarrow[E]{\varepsilon^*} q'$ Here q' is ε-reachable from q
4. $q \xrightarrow[E^*]{a} q'$ ε & one char
5. $q \xrightarrow[E^*]{w} q'$ $w \neq \varepsilon$



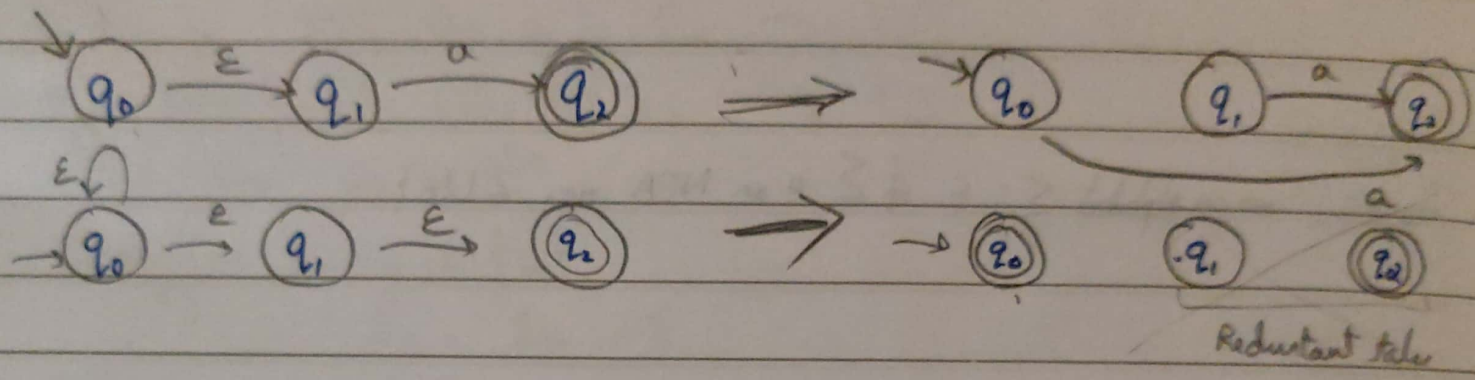
$1 \xrightarrow[E]{a} 3$ $\quad$ $1 \xrightarrow[E^*]{a} 3$ $\quad$ $1 \xrightarrow[E]{\varepsilon^*} 1$

$\xrightarrow[E]{\varepsilon^*}$ $\quad$ $\xrightarrow[E^*]{w}$ $\quad$ $\xrightarrow[E^*]{ab}$

$\xrightarrow[E]{\varepsilon}$ $\quad$ $\xrightarrow[E^*]{a}$

$1 \xrightarrow[E^*]{b} 3$

Converting ε - NFA to NFA (Example



Converting ε - NFA to NFA

Consider an ε NFA $E = (Q_E, i_E, \Sigma \cup \{0\}, \xrightarrow{\varepsilon}, F_E)$

Construct an NFA $N = (Q_N, i_N, \Sigma, \xrightarrow{N}, F_N)$

where $Q_N = Q_E$, $i_N = i_E$

$$q \xrightarrow[N]{a} q' \overset{\text{def}}{=} q \xrightarrow[E^*]{a} q'$$

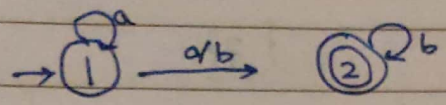
$$F_N = \begin{cases} F_E \cup \{i_E\} & \text{if } \{q \mid i_E \xrightarrow[\varepsilon]{\varepsilon^*} q\} \cap F_E \neq \emptyset \\ F_E & \end{cases}$$

Q_N

TUTORIAL

1. 2 State NFA to DFA conversion

NFA :



DFA : States = {ϕ}, {1}, {2}, {1,2}

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

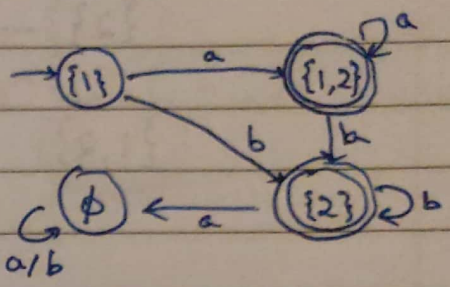
δ_D : $\{1\} \xrightarrow{a} \{1,2\}$

$\{1\} \xrightarrow{b} \{2\}$

$\{2\} \xrightarrow{b} \{2\}$

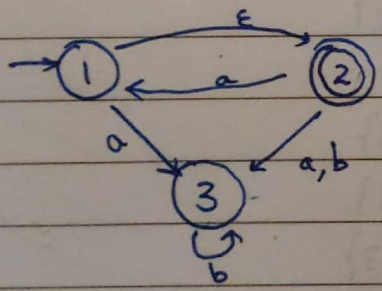
$\{1,2\} \xrightarrow{a} \{1,2\}$

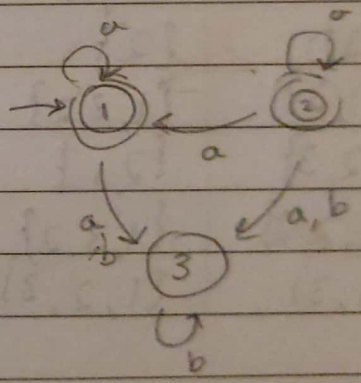
$\{1,2\} \xrightarrow{b} \{2\}$



2. 3 State ε-NFA to DFA conversion

εNFA :





Type ~~of Relation~~ 1. $q \xrightarrow{\varepsilon} q'$

2. $q \xrightarrow{a} q'$

3. $q \xrightarrow{\varepsilon^*} q$

4. $q \xrightarrow{a} q'$

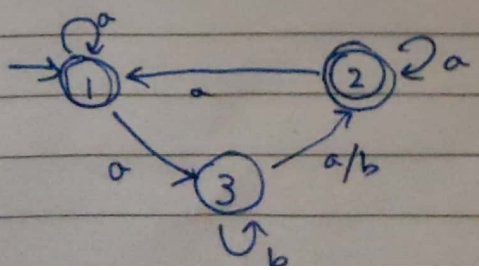
5. $q \xrightarrow{a} q'$

~~(1,2)~~ $1 \xrightarrow{\varepsilon} 2$

~~(2,1)~~ ~~(1,3)~~ ~~(3,3)~~ ~~(2,3)~~ $2 \xrightarrow{a} 1$

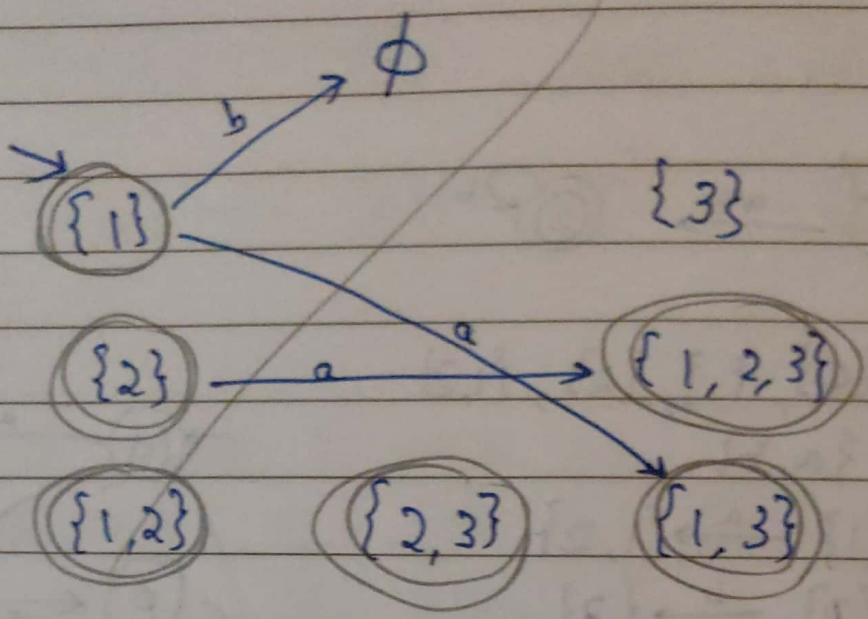
$1 \xrightarrow{b} 3$ $2 \xrightarrow{a} 2$ (2)

NFA :



δNFA	ε	a	b
→ 1	2	3	-
2	ϕ	1,3	3
3	-	-	3

DFA



	a	b	Reachable from {1}?
{ϕ}	{ϕ}	{ϕ}	
{1}	{1,3}	{ϕ}	✓
{2}	{2,1}	{ϕ}	
{3}	{2}	{3,2}	
{1,2}	{1,3,2}	{ϕ}	✓
{2,3}	{2,1}	{2,3}	✓
{1,2,3}	{1,3,2}	{2,3}	✓
{1,3}	{1,2,3}	{2,3}	✓

Direct Method

REGULAR LANGUAGES

L is regular if L is recognized by some DFA D.

A language with no strings in it is denoted by $0 = \phi = \{\phi\}$

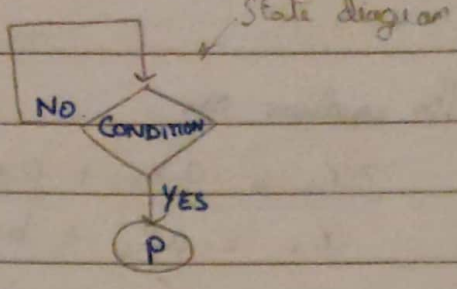
A lang. ε is denoted by $1 = \{\varepsilon\}$

Union of two languages L_1 & L_2 is denoted as $L_1 \cup L_2 = \{\omega \in \Sigma^* \mid \omega \in L_1 \vee \omega \in L_2\}$

Concatination

Concatination of 2 languages L_1 & L_2 is denoted as $L_1 . L_2 = \{\omega \in \Sigma^* \mid \omega_1 \in L_1 \wedge \omega_2 \in L_2 : \omega = \omega_1 \omega_2\}$

Equation as Automaton

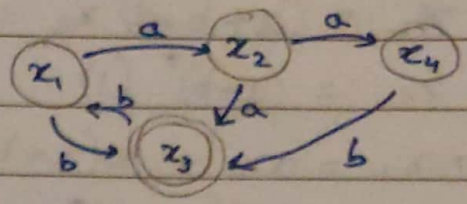


A condition can be considered as an automaton

CONDITION = NO. condition + YES . P

The i/p is NO, NO .... YES.p

Consider an automaton as in figure.



Suppose we start at x_1,

$$x_1 = ax_2 + bx_3$$

From every state we can go to other states via i/p & can be written as an equation.

The equations are: $x_1 = ax_2 + bx_3$ (Start state is insignificant, as we look how to reach ϵ from every state)

regular expressions, $x_2 = ax_3 + ax_4$

$x_3 = bx_1 + \epsilon$ (Since x_3 is the final state)

$x_4 = bx_3$

i.e. If x_2 is mapped to some language, then $x_1 = ax_2$ has the language ~~a~~ concatenated with that of x

$L = \epsilon L$

$L = 0$

ARDEN'S LEMMMA

If A & B are reg languages & A does not contain ϵ then, the equation $x = Ax + B$ has a unique solution $x = A^*B$ BA^*

EXAMPLE

Let $x_0 = B$ (First approximation)

$x_1 = Ax_0 + B = AB + B$

$x_2 = Ax_1 + B = A^2B + AB + B$

$\vdots$

$x_i = A^iB + A^{i-1}B + \dots + B$

$\vdots$

$n \to \infty \; x_n = A^*B$ (infinite language)

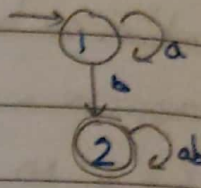
$\therefore A^*B = A(A^*B) + B$

$= A^*B + B$

$= (A^* + 1)B$

$= A^*B$

Example in book



The equations are

$\rightarrow x_1 = ax_1 + bx_2$ (1)

$x_2 = ax_2 + bx_1 + 1$ (2)

$x_2 = (a+b)x_2 + 1$

$\therefore x_2 = (a+b)^*$ (1) [Arden's lemma from (2)] (3)

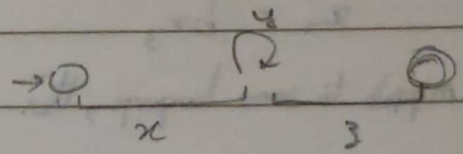
$\rightarrow x_1 = ax_1 + b(a+b)^*$ (4) [(1) & (3)]

$\therefore x_1 = a^*b(a+b)^*$ [Arden's lemma from (4)] (5)

PUMPING LEMMA

$\forall$ language over Σ i.e. regular, $\exists p \geq 1$ (regular (L)), $\forall s \in L$, $\exists x,y,z \in \Sigma^*$ (then are 3 substrings such that

1) $s = xyz$
2) $y \neq \varepsilon$
3) $|xy| \leq p$
4) $\forall i \quad xy^iz \in L$



APPLICATION OF PUMPING LEMMA

R(L) = L is a regular language, R(L) $\Rightarrow$ P(L) means L satisfies pumping lemma $\neg P(L) \Rightarrow \neg R(L)$

Check if L is a regular language

Check if L is a regular language

1. $L = \{ a^n b^n \mid n \in \mathbb{N} \}$ $\quad \Sigma = \{a, b\}$

oof 1) Assume L is regular
2) L satisfies Pumping property
3) Let s be an arbitrary string in L
4) $s = xyz$
5) Cases
5) y falls completely in a^n
   a) $y = a^k$
   b) $x = a^m$
   c) $z = a^{n-(k+m)} b^n$
   d) $xz \notin L$
6) But $xy^0z \in L$ by PL

$xyz = \overbrace{aaaaa\ldots a}^{n}\overbrace{bb\ldots b}^{n}$ (x: a^m, y: a^k, z: $a^{n-(k+m)}b^n$)

$xz = a^m a^{n-(k+m)} b^n$
$= a^{n-k} b^n \notin L$

6) ∴ Contradiction 4, 5

ASSUMPTION
1), pumping lemma
2), pumping lemma
3,2) Pumping lemma (a)

$0 < k \leq n$
$0 \leq m < n$

$a^{n-k}\ a^{k}\ b^{m}\ a^{k}\ b^{m}\ b^{n-m}$

5.2) y falls completely in b^n (left as exercise)

5.3) y straddles between as & bs

a) $y = a^k b^m$ for some $0 < k \leq n$ $0 < m < n$ $xyz = \underbrace{aaa....aa}_{n}\underbrace{bb...b}_{n}$ (x | y | z)

b) Taking $i=2$, $xy^2z \notin L$

c) But $xy^2z \in L$ by PL

d) ∴ Contradiction