DATE: / /
PAGE NO.:

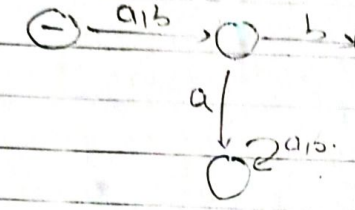
Ch 7 Kleene's Theorem (contd.)

Converting RE's to FA's. (contd.)

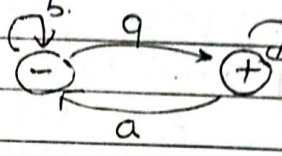
Rule 3 - FA_1 accepts r_1, FA_2 - r_2, FA_3 accepts $r_1 r_2$.

Proof - Strings with 1st part as r_1 & 2nd part as r_2.

ex. FA_1 - words with b as 2nd letter.



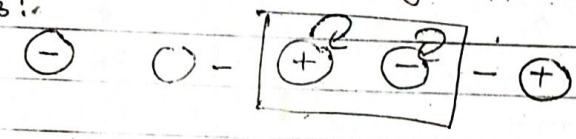
FA_2: odd a's.



Str. abbabaa - word in prod. lang. $L_1L_2 \rightarrow (ab)(abbaa)$

start on FA_1, reach + after 2nd letter b. Somehow jump into FA_2, run the remaining i/p; abbaa, reach +.

FA_3:



But this does not work.

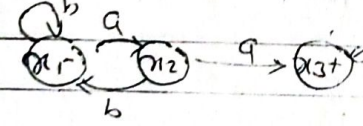
ex. $abababb = (abab)(bab)$.

On FA_1 we reach +, when we read 2nd b. But we have not finished on FA_1. If we start from FA_2, we don't reach +, because abbab has even a's.

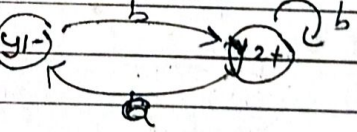
Here after we reach + on FA_1, we must continue to run the string on FA_1 for 2 more letters. Then jump on FA_2.

Problem: How do we know when to jump from FA_1 to FA_2?

Sol:- FA_1 - accepts aa

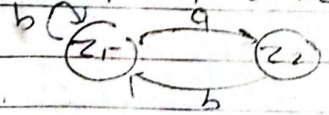


FA_2 - ends in b.



Start with $S_1 \cdot z_1$, which is exactly like x_1. On b, return to z_1. On a, go to z_2 - like x_2. From z_2, on b, go back to z_1.

$z_1 = x_1$, $z_2 = x_2$



From z_2, on a, go to z_3 which is somewhat like x_3.

x_3 - dual identity. i) final state (end of i/p)

ii) continue at x_3 for rest of i/p.

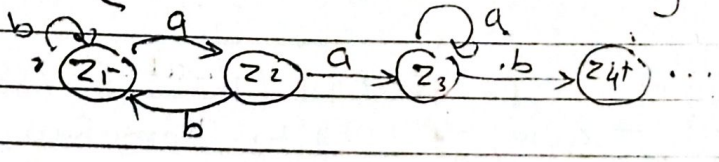
$\therefore z_3 = \begin{cases} x_3, \text{ we're still on } FA_1 \text{ or} \\ y_1, \text{ we've begun on } FA_2. \end{cases}$

In z_3, on a, 3 interpretations:
- we are back in x_3 on FA_1, or
- we have just finished on FA_1 & beginning to run FA_2
- or we've looped from y_1 to y_1 on FA_2

$= x_3$ or $y_1 = z_3$.

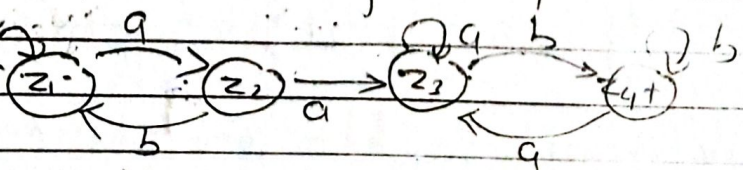
i.e. in z_3 on a, loop back to z_3.
in z_3 on b, go to z_4

final $\uparrow$ $+z_4 =$ { in x_3 on FA_1 or finished FA_1 & now in y_1 or in y_2 via y_1 } $= x_3$ or y_1 or y_2

FA3 : 

In z_4, on a, choices:
{ in x_3 on FA_1 or finished FA_1 & begin y_1 or from y_2 back to y_1 } $= x_3$ or $y_1 = z_3$

i.e. in z_4 on a, go to z_3.
In z_4 on b,
in x_3 or begin y_1 or loop y_2 to y_2 $= x_3$ or y_1 or y_2 $= z_4$

$\therefore$ FA3 : 

Thus without double a we never reach to z_2 & we end in z_4 if whole word ends in b.

Algorithm : for forming mlc FA_3 :
Make a z-state for every non-final x state in FA_1 reached before ever hitting a final state on FA_1.
For each final state in FA_1, we establish a z-state that expresses the options that we are continuing on FA_1 or are beginning on FA_2.

DATE: / /
PAGE NO.:

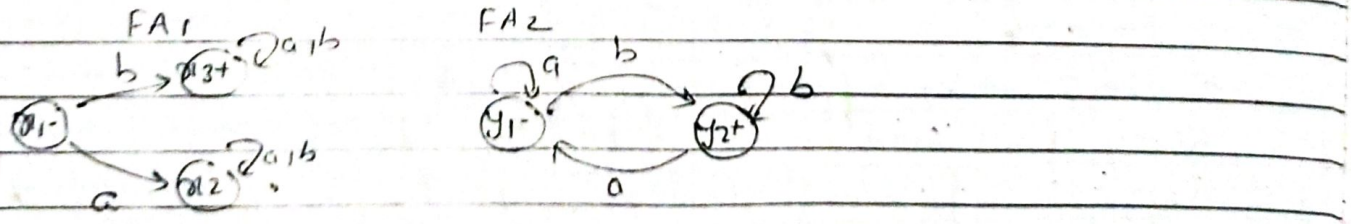
Are in $x_{something}$, which is a + state but still continuing on FA1
or
have finished the FA1 part of i/p string & have jumped to y_1 to commence tracing the remainder of i/p on FA2

After we have reached a jump-to-FA2 state, any other state we reach has an x & a y possibility like the z-states in the union m/c. a whole set of possible y-states

Full nature of a z-state is
{ are in $x_{something}$ continuing on FA1 or
are in a set of $y_{something}$ conti. on FA2.

There are only finitely many possibilities for such z-states, so FA3 is a finite m/c.
Thus FA3 accepts only strings that first reach a final state on FA1, jump to y_1 & then reach a final state on FA2.
To be in FA1, FA2 means to end in a final state in FA2, so any z-state is a final state if it contains a y-machine final state as a possible position for i/p.

Ex Construct m/c for product of long. L1, all words that start with a b & L2, all words end with a b.



→ let $z_1 = x_1$ on a, go to x_2, let $z_2 = x_2$
on b, " x_3, which being a final state we have option of jumping to y_1.
$z_3 = x_3$ or y_1.
From z_2, like x_2, both a & b take us to z_2.

DATE: / /
PAGE NO.:

$Z_3 = y_1$ or x_2

from Z_2, @ b will take us from y_2 to y_2, y_2 to y_2 & jump to x_1, x_1 to x_3.

$Z_4 = y_2$ or x_1 or x_3.

from Z_3, a will take us from y_1 to y_1 or x_2 to x_2.
Z_3 has an a-loop.
from Z_3, b will take us from y_1 to y_2 or y_1 to y_2 to x_1, or x_2 to x_2.

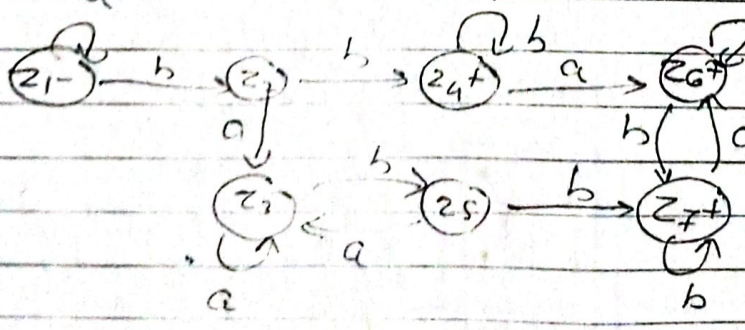
$Z_5 = y_2$ or x_1 or x_2.

from Z_4, a → y_2 to y_1, x_1 to x_2, x_3 to x_3.
Z_6. y_1 or x_2 or x_3.
Z_4, b → y_2 to y_2, y_2 to y_2 to x_1, x_1 to x_3, x_3 to x_3.
$= Z_4$

Z_5, a → y_2 to y_1, x_1 to x_2, x_2 to x_2, $= Z_3$.
Z_5, b → y_1 to y_2, y_2 to y_2 to x_1, x_1 to x_3, x_3 to x_2
$Z_7 = y_2$ or x_1 or x_2 or x_3.

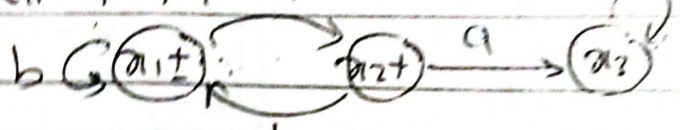
Z_6, a → to Z_6 for each of its 3 components.
b → y_1 to y_2, y_1 to y_2 to x_1, x_2 to x_2, x_3 to x_3
$= Z_7$.

Z_7, a → y_1 to y_1, x_1 to x_2, x_2 to x_2, x_3 to x_3 $= Z_6$.
Z_7, b → y_2 to y_2, y_2 to y_2 to x_1, x_1 to x_3, x_2 to x_2, x_3 to x_3 $= Z_7$.

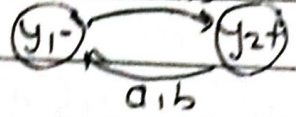


The only final states are those that contain possibility of x_3. M/c accepts words with bb, but it is not efficient.

ex. FA1 : a

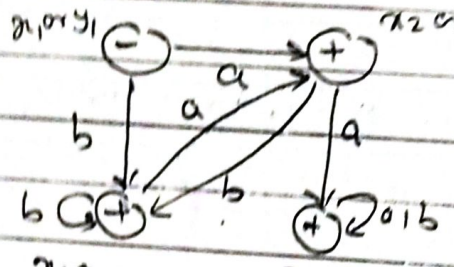


→ words not contain aa

FA2: 

— words with odd no of letters

FA3: L1L2:



all states except — are + — is left the instance ith letter is read & it can never be reentered.
mlr accepts all words except Λ

FA3: product lang L1 L2 because if word w has odd letters, factor it as (Λ)(w), Λ is in L1 & w in L2.
If it is even (not 0),

w = (1st letter)(rest)
must be in L1 (can't contain aa) — in L2.

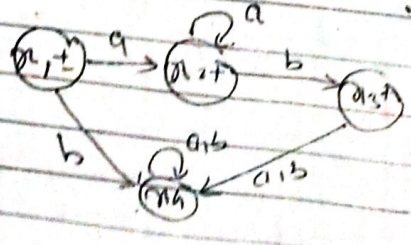
Only word Λ can't be factored into a part in L1 & a part in L2

Rule 4 - r is RE, FA1 accepts lang. defined by r, then there is FA2 that will accept lang. def'd by r*.

Proof - r* contain Λ → ∴ start st = final st

ex: RE. r = a* + aa*b. → str. of only a's & str. of some a's ending in b.

closure (a* + aa*b)* — words in which each b has a on left: all str. without a double b that do not begin with b.

FA1 for r*: 

DATE: / /
PAGE NO:

FA2 for r^*:

$+z_1 = r_1$

In z_1 on b, $+r_4 = z_2$

z_1 on a, goto z_3

$+z_3 =$ { r_2 & continue or
accepted a section & go back to r_1

$+z_3 = r_2 + r_1$

While we are scanning i/p string, we arrive at a break between one factor of type r & another factor, first ends at + & 2nd should begin at −.

However a factor does not have to stop at first +. It may terminate at 4^{th} +. So this jump is only an option.

In case of product of 2 m/c's, when we hit a + on 1st m/c, we can continue on that m/c or jump to − on 2nd m/c. Here when we hit a +, we can also jump back to − (on same m/c) or ignore + status & continue or end completely.

In z_2, i/p is rejected, so stay at z_2.

In z_3, on b, if r_2 cond^n, goto r_3
if r_1 " " r_4

but from r_3, we can continue or
start from scratch at r_1

$\therefore +z_4 = r_1$ or r_3 or r_4.

r_4 is final state.

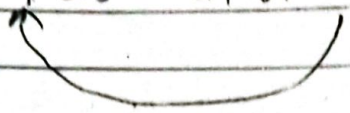
In z_3, on a, if r_2, r_2,
if r_1, $r_2 \rightarrow r_1$

$\therefore$ we return to z_3.

In z_4, on b, from r_1, r_3, r_4 we go to r_4

In z_4, on a, if x_1 goto x_2
if x_3 goto x_4
if x_4 " x_4

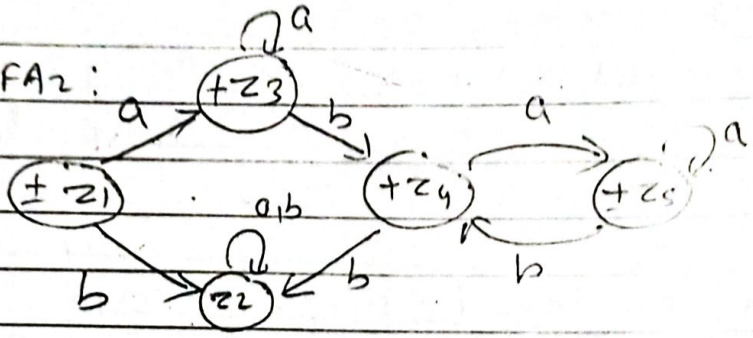
$+z_5 = x_1$ or x_2 or x_4



z_5, on a, get to x_1 or x_2 or x_4 : z_5
b " x_1 x_3 x_4 = z_4

$(a^* + aa^*b)^*$

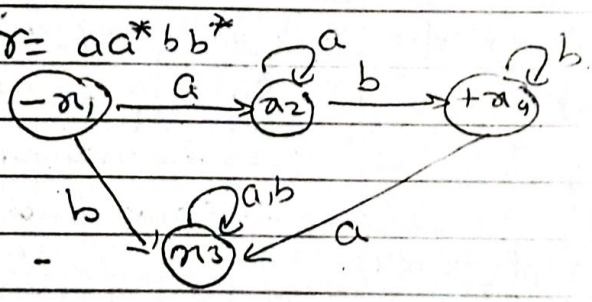
∴ FA2 :



---

eg. $r = aa^*bb^*$

FA



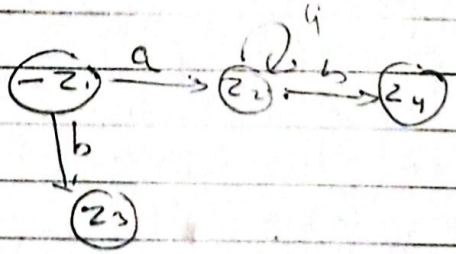
$r^* = (aa^*bb^*)^*$.

FA for r^* :

$-z_1 = x_1$

$a \rightarrow z_2 = x_2$

$b \rightarrow z_3 = x_3$.



$z_2 \rightarrow a \rightarrow z_2$

$b \rightarrow z_4$ $\quad +z_4 = x_1$ or x_4.

z_4, $a \rightarrow x_3$ (if in x_4)

x_2 (if in x_1)

$z_5 = x_2$ or x_3. ⟶ (optional)

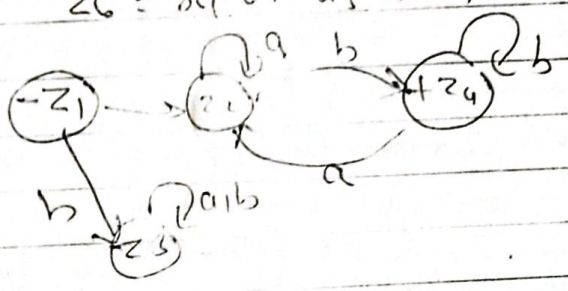
but x_3 is worthless.

DATE: / /
PAGE NO.:

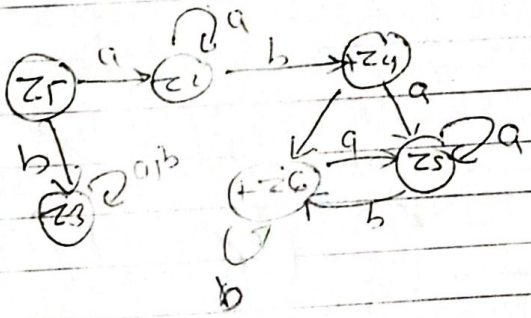
$z_4, b \to x_4$ (if in x_4)

x_3 (x_1)

$z_6 = x_1$ or x_3 or x_4. (optional)



or



This m/c does not accept Λ.

$\because z_1 = x_1$ is not correct true only when x_1 is also a final state. Then only z_1 can be a final state.

$\therefore$ we need 2 states like x_1 one - x_1 & non final & other x_1 & final.

Reason for nonfinal FA $\to$ for i/p, we may be required to reenter state x_1 many times.

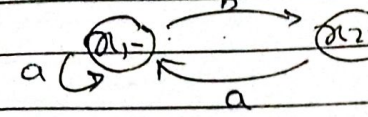
If x_1 is nonfinal in FA_1, but we convert it to z_1 which is final, then when an i/p ends in x_1 on FA_1 & is not accepted on FA_1, we don't want to say that it ends in z_1 which causes it to be accepted on FA_2.

Above m/c has no problem because x_1 on FA_1 can never be reentered (no incoming edges)

$\therefore$ here $z_1 = x_1$

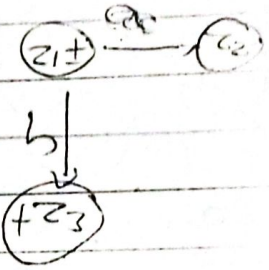
$\therefore$ z_1

ex. FA: words with odd b's.



z_1 = x_1 & final state

z_2 = x_1 & non " "



$z_1, b \to x_2$

$\therefore$ x_1 or $x_2 = z_3$

$z_2 \to a \to z_2$

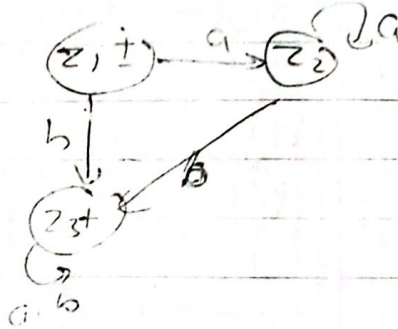
$b \to z_3$

$z_3 \to a \to z_3$ if in x_1, go in x_1

" x_2, " x_2

$b \to z_3 \to a_1 \to x_2$

$a_2 \to x_1$



Conclusion of Kleene's Th^m :-

All RE's have corresponding FA. While we build RE from elementary building blocks by recursive def^n, we can simultaneously build corres. FA from 4 preceding algorithms. This is a powerful example of the strength of recursive def^n

ex: To find FA for RE $(ab+a^*)$.

RE can be built by repeated application of rules: letter, sum, product, star.

Process of expression & m/c building:

a is letter — FA_1 accepts it.

b " FA_2 "

ab is lang. of product of 2 m/c's — $FA_3 = FA_1\ FA_2$

a^* is lang. of closure of m/c FA_1 — FA_3 accepts it

$ab+a^*$ is " sum of FA_2 & FA_3 — FA_4 accepts it.

done.

DATE: / /
PAGE NO.:

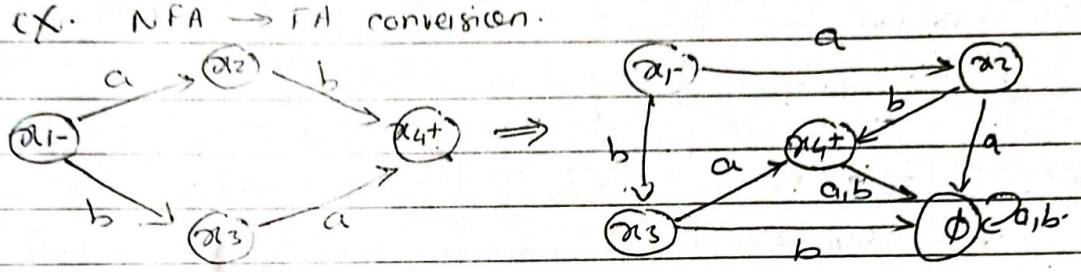
Thus every lang. accepted by an FA can be accepted by TG, " " " TG " defnd by RE & " defnd by RE can be accepted by FA.
∴ RE, FA, TG are eqvt.

TG's seem more understandable, we use it often instead of FA's (having to specify what happens in every state to every letter).

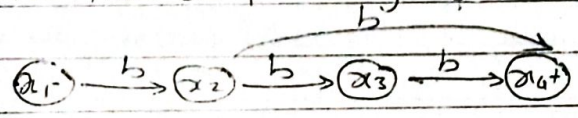
TG's are not more powerful than FA's, there are no extra lang. that TG's can accept that FA's could not handle already.
There are some lang. that FA's can't accept & we need more powerful type of m/c than a TG to deal with them.

DATE: / /
PAGE NO.:

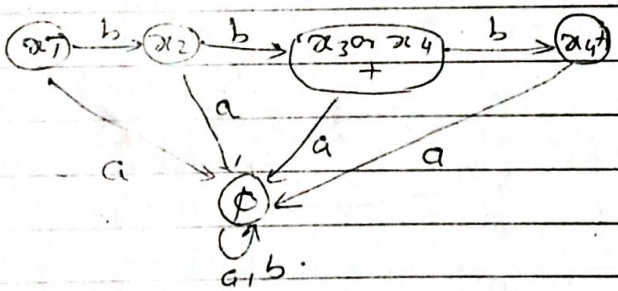
ex. NFA → FA conversion.



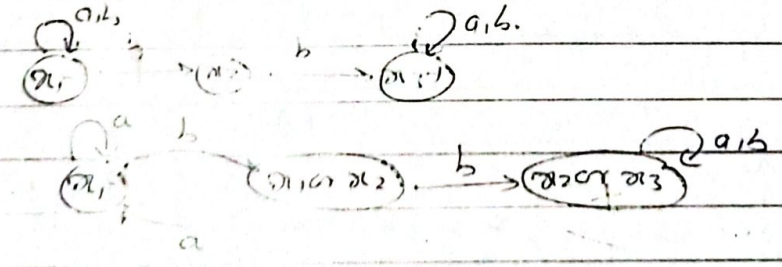
ex. NFA that accepts lang. {bb bbb}



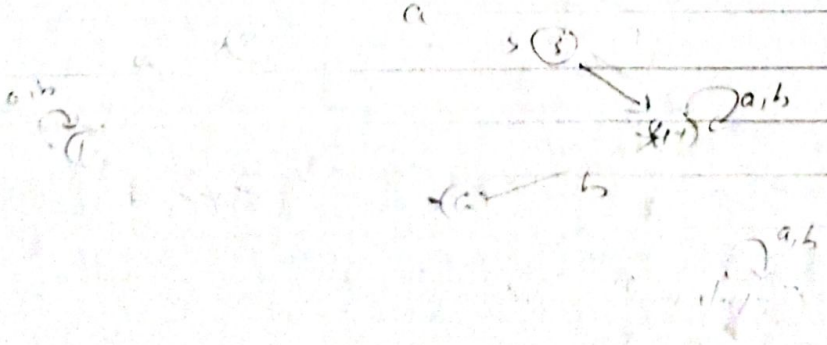
FA:



ex. NFA - i/p: will have bb in them.



ex. NFA i/p: a,b; triple letter a



ex. construct NFA for RE. $01^* + 1$

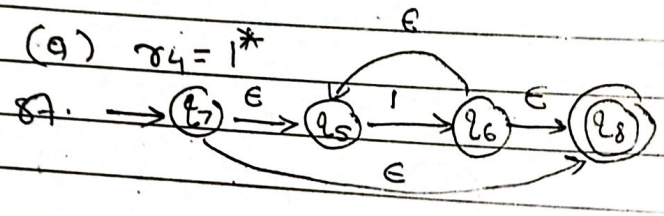
→ $01^* + 1 = (0(1^*)) + 1$

$= r_1 + r_2$

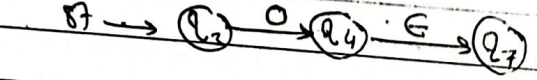
$= * (r_3 r_4) + r_2$

$r_3 = 0, \; r_4 = 1^*$

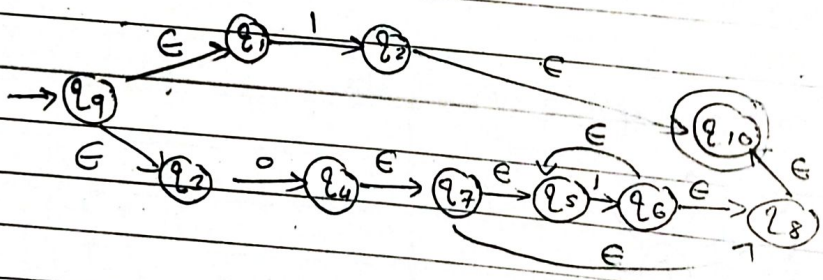
(a) $r_4 = 1^*$



(b) $r_1 = 01^*$



(c) $r = 01^* + 1$.



→ Power of FA & NFA is same -

L = {00, 01}

DATE: / /
PAGE NO.:

Ch. 7 Kleene's Theorem (contd.)

Nondeterministic Finite Automata (NFA)

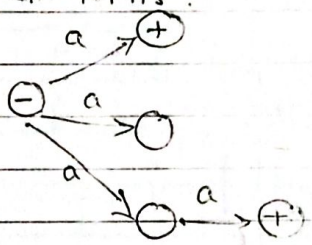
m/c that occurs in practice more frequently than TG, but that shares with it the property of being nondeterministic.

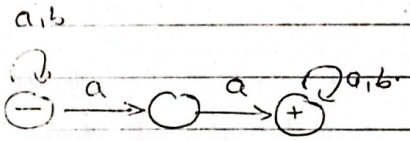
✓ Def^n NFA is a TG with a unique start state with the property that each of its edge labels is a single alphabet letter.

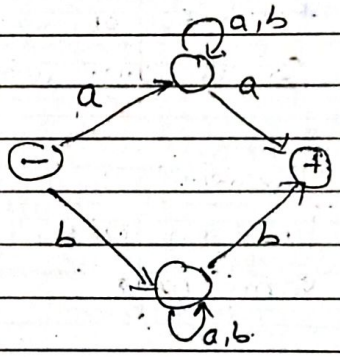
Nb: [Regular deterministic finite automata are referred to as DFA].

✓ Scope of FA is expanded by allowing arbitrarily many a & b edges coming out of each state.
i/p str. is accepted if there exists any one possible path to +.

ex. NFAs:



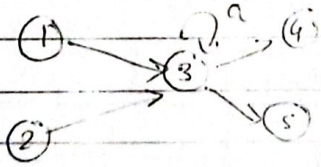


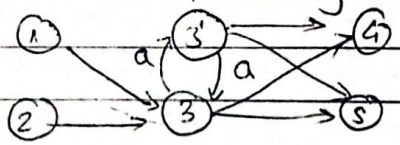


NFAs were invented by Rabin & Scott in 1959.

ex. one possible use of NFA is to eliminate all loop states in a given FA.

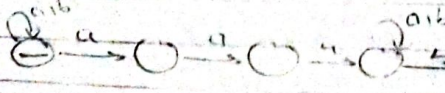
Use twin for 3, 3' & jump bet^n 3 & 3'.





adv. the fact of whether looping in state 3 occurs for given i/p, is recorded in whether the path followed by i/p goes thro' state 3' or not. In comp^r prog., state 3' may set a flag alerting one to the incidence of looping.

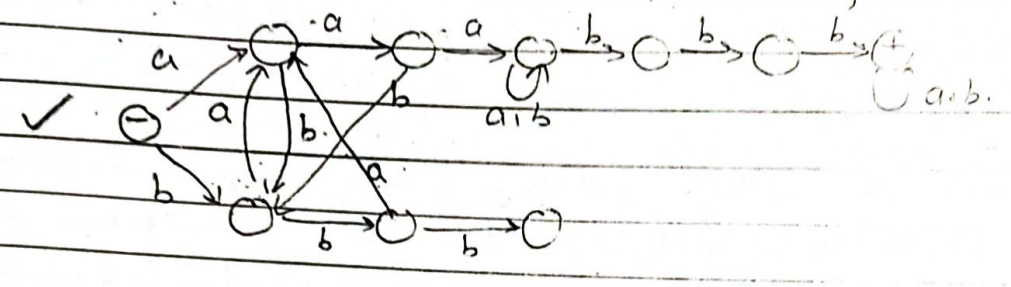
ex NFA accepts str. with a triple a followed by a triple b.



Here some bbb can occur before the first aaa (by looping

at —) & then has another bbb later

1b first triple b has to be preceded by a triple a,



Because NFA is a type of TG & TG can be converted into FAs by kleene's th^m, i.e. all NFA's can be converted into FAs that accept same lang.

all FAs can be considered NFAs that do not make use of the option of extra freedom of edge production.

So as long acceptors, NFA = FA.

Theorem 7 →

For every NFA, there is some FA that accepts exactly the same lang.

Proof 1 - convert NFA into RE by state bypass oper?
From RE, construct FA. using kleene's th^m

Proof 2 -

Power of FA & NFA

DATE: / /
PAGE NO.:

NFA to DFA conversion -

ex. NFA state table :

Q \ Σ	0	1
q_0	$\{q_0, q_1\}$	$\{q_1\}$
q_1	ϕ	$\{q_0, q_1\}$

$M = (\{q_0, q_1\}, \{0,1\}, \delta, q_0, \{q_1\})$.

DFA: $M' = (Q', \Sigma, \delta', [q_0], F')$.

$2^Q = \{\phi, \{q_0\}, \{q_1\}, \{q_0, q_1\}\}$.

$Q' = 2^Q$, use [ ] to denote states in DFA.

$\therefore Q' = \{[q_0], [q_1], [q_0, q_1]\}$.

ϕ doesn't denote any states of given NFA.

Find δ' i.e. transitions from every state Q' on both 0 & 1.

from state table of given NFA we can write.

$\delta'([q_0], 0) = [q_0, q_1]$

$\delta'([q_0], 1) = [q_1]$

$\delta'([q_1], 0) = \phi$

$1 = [q_0, q_1]$

$\delta'([q_0, q_1], 0) = [\delta(q_0, 0) \cup \delta(q_1, 0)]$

$= [\{q_0, q_1\} \cup \phi] = [q_0, q_1]$

$\delta'[[q_0, q_1], 1) = [\delta(q_0, 1) \cup \delta(q_1, 1)]$

$= [\{q_1\} \cup \{q_0, q_1\}]$

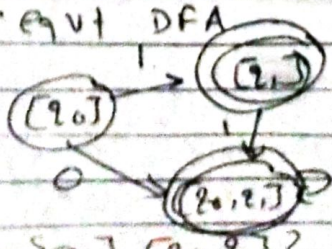
$= [\{q_0, q_1\}]$

$= [q_0, q_1]$

DFA state table

δ' \ Σ	0	1
$[q_0]$	$[q_0, q_1]$	$[q_1]$
$[q_1]$	–	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$

eqvt DFA



Relabel state symbols :

A : [q_0] , B : [q_1], C : [q_0, q_1].

DFA :

δ' : $Q' \backslash \Sigma$	0	1
A	C	B
B	-	C
C	C	C

$Q' = \{A, B, C\}$

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

$q_0 = A$

$F' = \{B, C\}$.

NFA with ϵ-moves/transitions –

$\delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$.

ex → (q_0) —ϵ→ (q_1) —ϵ→ ((q_2)) 2

Str with any no of 0's followed by " 1's " " 2's.

Difficult to draw FA, NFA.

State table for above NFA with ϵ-moves

$Q \backslash \Sigma \cup \{\epsilon\}$	0	1	2	ϵ
q_0	$\{q_0\}$	ϕ	ϕ	$\{q_1\}$
q_1	ϕ	$\{q_1\}$	ϕ	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$	ϕ

Pls. refer to xerox pages for –
Conversion from NFA with ϵ-moves to NFA w/o ϵ-moves & to DFA.