Theory of Automata

Languages

Formal ← → Informal

Syntactic — Semantic

↳ structure

↳ logic focused

ALGOL : has 113 letters

automata: something that works automatically

Empty/NULL string (λ/Λ)

Words

↳ strings belonging to language

↳ if ✓

↳ iff ✗

$L = \{x^n : n = 1, 2, 3, \dots\}$

x^n: concatenation

Strings

concatenation of finite symbols from alphabet

$\Sigma = \{a, b\} \rightarrow a, ab, aaab \dots$

Alphabets (Σ)

↳ finite non empty set of symbols

↳ letters, digits, GOTO, IF, operators

↳ $\Sigma = \{a, b\}$

Length of string ($|s|$)

↳ no of letters

$\Sigma = \{B, aB, bab, d\}$

$s = B\ aB\ bab\ B\ d$

$|s| = 5$

Defining alphabet rule

no letter should be prefix of another

$\Sigma_1 = \{B, aB, bab, d\}$

$\Sigma_2 = \{B, Ba, bab, d\}$ → invalid

invalid as both start with same element

Reverse of string (Rev(s)/s^r)

↳ reverse words order → not letter

$s = BaBbabBd$

$Rev(s) = d\ B\ bab\ aB\ B$

Concatenations

↳ $w^n = \overbrace{ww\dots w}^{n}$

$abba^2 = abbaabba$

↳ $w^0 = \lambda$ → NULL/EMPTY string

$abba^0 = \lambda$

* Operation

↳ all possible strings from alphabet

↳ all possible combinations

↳ includes null too

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

$\Sigma^* = \{\lambda, a, b, aa, bb, ab, ba \dots\}$

+ Operation

↳ all possible combinations

↳ doesnt includes null too

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

$\Sigma^+ = \{a, b, aa, bb, ab, ba \dots\}$

Language

alphabet ↑

any subset of Σ^*

$\{\lambda\}$ → diff language

$\{a, aa, aab\}$ → diff language

words

order doesnt matter in sets

Sets

$\emptyset = \{\} \neq \{\lambda\}$

$\emptyset$: set size = 0

$\{\}$: set size = 1

λ: string length = 0

all words are strings

not all strings are words

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

n: exactly n

≥ 0: infinite language

can be zero so null string include

a^n: all a's before b's

language where a and b are equal

Set operations

$A = \{a, ab, aaaa\}$, $B = \{bb, ab\}$

- Union '$\cup$' = combines
  - $A \cup B = \{a, ab, aaaa, bb\}$
- Intersection '$\cap$' = common
  - $A \cap B = \{ab\}$
- complement '$\overline{set}$' = opposite
  - $\overline{A} = \Sigma^* - A$
  - $\{\overline{a, ba}\} = \lambda, b, ab, aa, bb, aaa\}$
- Difference 'setA − setB' = remove all elements of setB from set A
  - $A - B = \{a, aaaa\}$

Star closure (Kleene)

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

Positive closure

$L^+ = L^0 \cup L^1 \cup L^2 \ldots$

absent include λ

Reverse of a language

- reverse word by letter → not order
- $L^R = \{w^R : w \in L\}$, $\Sigma = \{a, b\}$

  $\{ab, aab, baba\}^R = \{ba, baa, abab\}$
- $L = \{a^n b^n : n \geq 0\}$, $L^R = \{b^n a^n : n \geq 0\}$
- $\Sigma = \{a, \underline{aB}, \underline{b}\}$

  $\{a, \underbrace{aBb}\}^R = \{a, baB\}$

  switch alphabets

Concactation of a language

- $L^n = LL \ldots L$

  $\{a, b\}^3 = \{a, b\}\{a, b\}\{a, b\} =$

  $aaa, aab, aba, abb, baa, bab, bba, bbb\}$
- $L^0 = \{\lambda\}$
- Special case:

  $\{a, bba\}^0 = \{\lambda\}$

Regular expressions : generator of language

- $(a+b.c)^* = \{\lambda, a, bc, abc, bca, aa, \ldots\}$
- $a^+ = a\underbrace{a^*}_{\text{will always come so no } \lambda} = \{a, aa, \ldots\}$
- $(ab)^* = \{\lambda, ab, abab, \ldots\}$ → can be used for even
- $(a^* b^*)$: b never comes first except when a = ?

  $\{\lambda, a, aa, b, bb, aab, abb\}$
- $(a+b)^* = \{\lambda, a, b, ba, ab, aa, bb \ldots\}$ → all a and b
- $(b^* a b^*)$: no order

  $\{a, bab, ba, ab, bbab\}$ → only 1 a exists
- (ab^*) : $\{a, ab, abb\}$ → only 1 a but at the start
- $(\underbrace{b^* a}_{\text{choose any time}} + \underbrace{ba^*}_{\text{since no star outside}})$: $\{a, b, bba, baa\}$
- $(ab)^*$: $\{\underbrace{ab}_{\text{length}} ab\}$
- $((a+b)(a+b))^*$: $\{\lambda, aa, ab, ba, bb\}$ → all possible even length

a	a	aa, aaaa
b	b	--
a	b	ab, abab
b	a	--

- $((a+b)^2)^* = \{\lambda, aa, bb, ab, ba, aaaa \ldots\}$ → all possible even length
- $(a+b)(\underbrace{(a+b)(a+b)}_{\text{even}})^* = \{a, b, aab \ldots\}$ → all possible odd length
- $(a+b)^* a (a+b)^* = \{a, ab, ba, bba \ldots\}$ → atleas 1 a
- $(a+b)^3 = (a+b)(a+b)(a+b) \cdot \{aaa, bbb, abb\}$ → exactly 3 length

Regular expression in UNIX

- [digit] = [0-9]
- uppercase letters = [A-Z]
- [alpha] = [A-Z a-z]
- [alnum] = [A-Z a-z 0-9]

8) 123 A Main St

RE = [0-9]$^+$[A-Z]?[A-Z][a-z]* [A-Z][a-z]*

what is question mark

imagine * as a loop

Finite automata (FA) : detector of language

A = { Q , Σ , δ , q_0 , F } → collection of 5 tuples

Q finite set of all states

Σ finite set of symbols, called alphabets → inputs

δ transition function

q_0 initial state

F final state → accepting state

Σ = { λ , l , m , lm } (λ + l + m + lm) cuz finite state

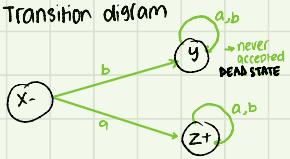
↳ automata starting with a

↳ Transitions table

old state	new state read a	new state read b
X-	z	y
y	y	y
z+	z	z

(states) → transition function (δ)

↳ Transition diagram



↳ regular expression

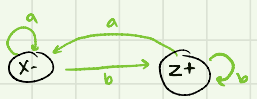
$a(a+b)^*$

↳ automata ending with b

↳ Transitions table

old state	new state read a	new state read b
X-	X	z
z+	X	z

↳ Transition diagram



↳ regular expression

$(a+b)^*b$

Q = {x, y, z}

Σ = {a, b}

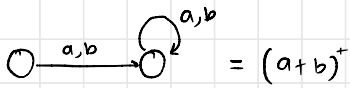
q_0 = X

F = Z

δ :

Transition Graph

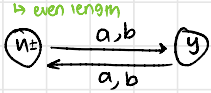
@ rigix



$= (a+b)^+$

	a	b
X±	y	y
y	x	x

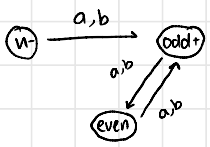
↳ ± accepts λ
↳ even length



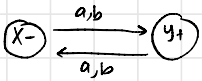
$((a+b)(a+b))^*$
even

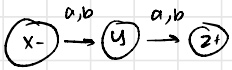
	a	b
X-	odd	odd
Odd+	even	even
even	odd	odd

↳ no ± as it accepts λ = even

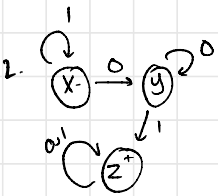


odd
$((a+b)(a+b))^*(a+b)$

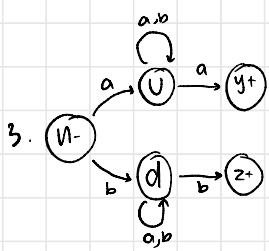


1. 

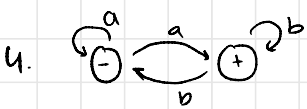
$(a+b)(a+b)(a+b)$
odd and 3 length

2. 

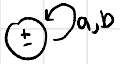
$(0+1)^* 01 (0+1)^*$
x 01 y

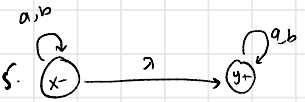
3. 

$a(a+b)^*a + b(a+b)^*b$
start end same letter

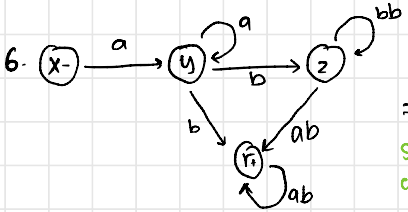
4. 

$(a+b)^*$



5. 

$= \Sigma^* \{a,b\}$ including λ

6. 

$= a^* (b(bb)^*)^* (ab)^+$
start a, end ab,
dumbs of odd bs

DFA DETERMINISTIC FINITE AUTOMATA

- 1 initial state
- final state ⩾ 0
- 1 transition of character 
- every possibility include dead states
- no null transition 
- unique next state 

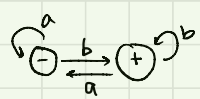
TG

- initial state >0
- final state >0 
- finite states, letters, transitions
- null transitions 

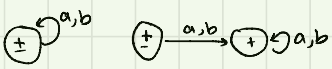
NFA NON DETERMINISTIC FINITE AUTOMATA

- 1 initial state
- final state ⩾ 0 
- multiple transition of characters
- not necessary every possibility dead states not necessary
- null transition 
- can be multiple next states 

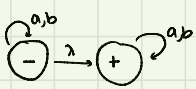
all possible a,b strings

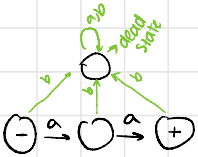


all possible a,b strings



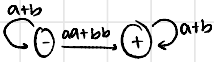
all possible a,b strings





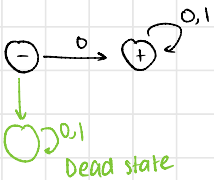
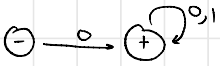
GTG

- TG
- regular expressions



NFA-Λ

if asked to make FA → NFA / DFA → usually

DFA	all string starting with 0	NFA
		

initial state → ○ (−)

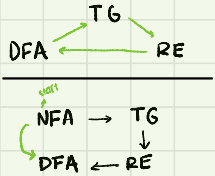
final state ◎ (+)

DFA → NFA

NFA → TG

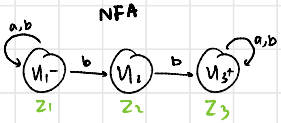
DFA → NFA-Λ

NFA-Λ → TG



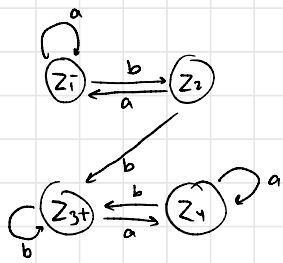
NFA TO DFA

NFA

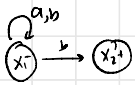


DFA

old states	a	b
$z_1^- \equiv \{x_1^-\}$	$z_1^- \equiv x_1$	$z_2 \equiv \{x_1, x_2\}$ (can put let it move)
$z_2 \equiv \{x_1, x_2\}$	$z_1 \equiv \{x_1, \phi\}$	$z_3^+ \equiv \{x_1, x_2, x_3\}$
$z_3^+ \equiv \{x_1, x_2, x_3\}$	$z_4^+ \equiv \{x_1, y_3\}$ (contains final state)	$z_3^+ \equiv \{x_1, x_2, y_3\}$
$z_4^+ \equiv \{x_1, x_3\}$	$z_4^+ \equiv \{x_1, x_3\}$	$z_3^+ \equiv \{x_1, x_2, y_3\}$

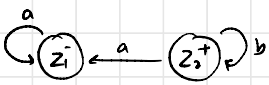


NFA



DFA

Old states	a	b
$Z_1^- \equiv x_1^-$	$Z_1^- \equiv x_1^-$	$Z_2^+ \equiv \{x_1^-, x_1^+\}$
$Z_2^+ \equiv \{x_1^-, x_1^+\}$	$Z_1^- \equiv x_1^-$	$Z_2^+ \equiv \{x_1^-, x_2^+\}$



EPSILON MISSING

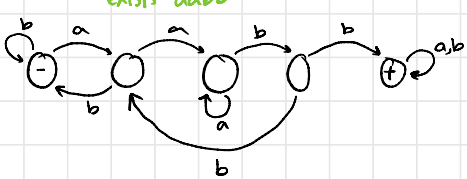
8) dosent have aabb DFA

↳ first make does have aabb

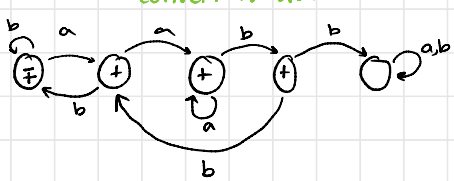
↳ convert final state to non final state

↳ convert non final states to final state

exists aabb



converting states



GENRALIZED TRANSITION GRAPH (GTG)

↳ initial state >0

↳ final state ≥0

↳ finite states, input letters

↳ Directed edges connecting regular expression states

NON DETERMINISM

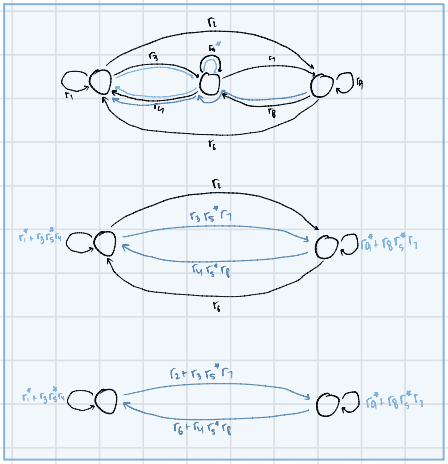
↳ TG, GTG

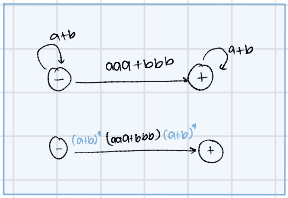
↳ There may exist none or more than one path for a certain string

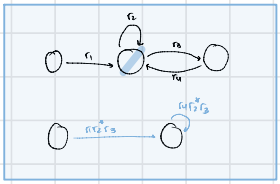
TG TO GTG CONVERSION

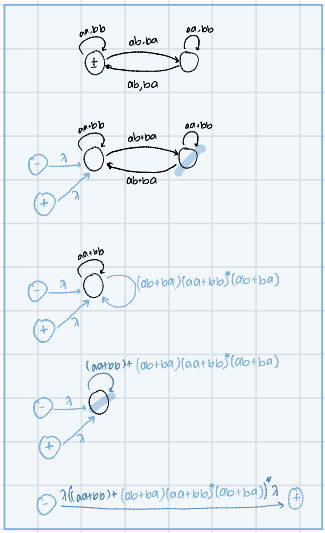
1. more than 1 start states = add new start state and connect old states with λ
2. more than 1 final states = add new final state and connect old states with λ
3. if a state has two or more edges = convert to 1 transition
   - ↳ loops = *
   - ↳ 2 possibilities = +
4. Bypass and state elimination
   - ↳ if 3 states connected in sequence : eliminate mid state and join the other two

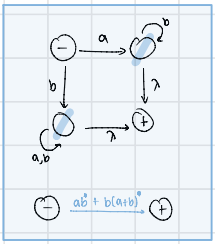
→ since start and final state same
→ and no other final state exists
add seperate start and final state and connect to old state

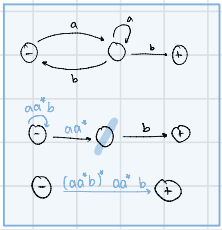










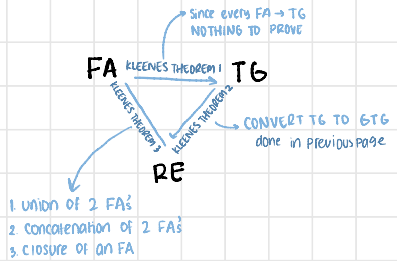


KLEENES THEOREMS

Regular Expressions

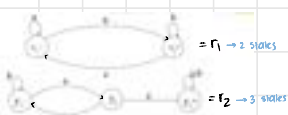
↳ FA
↳ TG
↳ RE

if can be expressed by one then the other and can be expressed aswell



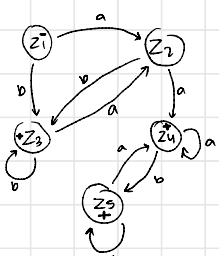
1. union of 2 FA's

↳ sum of 2 FA's = $r_1 + r_2$
↳ make transition table
↳ r_3 inital state = r_1, r_2 inital state
↳ r_3 final state = r_1, r_2 final state
↳ take $z_1 \equiv \{r_1 \text{ inihal}^-, r_2 \text{ initial}^-\}$
↳ r_3 final = r_1 and r_2 finals
↳ total no of possible states = states of r_1 x states of r_2



$r_3 = r_1 + r_2$ → 6 possible states

Old states	a	b
$z_1^- \equiv \{x_1^-, y_1^-\}$	$z_2 \equiv \{x_1^-, y_2\}$	$z_3 \equiv \{x_2^+, y_1^-\}$
$z_2 \equiv \{x_1^-, y_2\}$	$z_4 \equiv \{x_1^-, y_3^+\}$	$z_3 \equiv \{x_2^+, y_1^-\}$
$z_3^+ \equiv \{x_1^+, y_1^-\}$	$z_2 \equiv \{x_1^-, y_2\}$	$z_3 \equiv \{x_2^+, y_1^-\}$
$z_4^+ \equiv \{x_1^-, y_3^+\}$	$z_4 \equiv \{x_1^-, y_3^+\}$	$z_5 \equiv \{x_2^+, y_3^+\}$
$z_5^+ \equiv \{x_2^+, y_3^+\}$	$z_4 \equiv \{x_1^-, y_3^+\}$	$z_5 \equiv \{x_2^+, y_3^+\}$

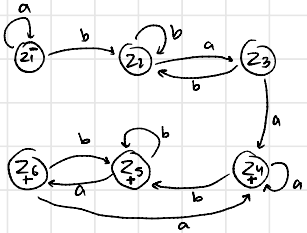


2. concatenation of 2 FA's

↳ $r_1 r_2$
↳ make transition table
↳ r_3 inhal state = r_1 inital state
↳ r_3 final state = r_2 final state
↳ r_1 final state = r_2 initial state
↳ take $z_1 \equiv \{r_1 \text{ inihal}^-\}$
↳ r_3 final = r_2 finals

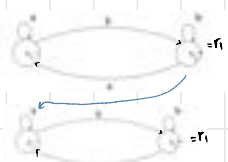
$r_3 = r_1 r_2$

Old states	a	b
$z_1^- \equiv \{x_1^-\}$	$z_1^- \equiv \{x_1^-\}$	$z_2 \equiv \{x_2^+, y_1^-\}$ (r_1 final, r_2 initial)
$z_2 \equiv \{x_2^+, y_1^-\}$	$z_3 \equiv \{x_1^-, y_2\}$	$z_2 \equiv \{x_2^+, y_1^-\}$
$z_3 \equiv \{x_1^-, y_2\}$	$z_4 \equiv \{x_1^-, y_3^+\}$	$z_2 \equiv \{x_2^+, y_1^-\}$
$z_4^+ \equiv \{x_1^-, y_3^+\}$	$z_4 \equiv \{x_1^-, y_3^+\}$	$z_5 \equiv \{x_2^+, y_1^-, y_3^+\}$
$z_5^+ \equiv \{x_2^+, y_1^-, y_3^+\}$	$z_6 \equiv \{x_1^-, y_2, y_3^+\}$	$z_5 \equiv \{x_2^+, y_1^-, y_3^+\}$
$z_6^+ \equiv \{x_1^-, y_2, y_3^+\}$	$z_4 \equiv \{x_1^-, y_3^+\}$	$z_5 \equiv \{x_2^+, y_1^-, y_3^+\}$



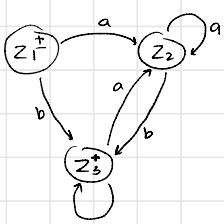
3. closure of an FA

↳ r^*
↳ make transition table
↳ r_3 inital = r_1 inital ±
↳ r_1 final state = r_1 initial state
↳ take $z_1 = \{r_1 \text{ inihal}^\pm\}$
↳ r_3 final = r_1 finals



$r_3 = r_1^*$

Old states	a	b
final → $z_1^\pm \equiv \{x_1^\pm\}$	$z_2 \equiv \{x_1^-\}$	$z_3 \equiv \{x_2^+, x_1^-\}$ (r_1 final, r_1 initial)
non final → $z_2 = \{x_1^-\}$	$z_1 = \{x_1^\pm\}$	$z_3^+ \equiv \{x_2^+, x_1^\pm\}$
$z_3^+ \equiv \{x_2^+, x_1^\pm\}$	$z_2 \{x_1^-\}$	$z_3^+ \equiv \{x_2^+, x_1^-\}$



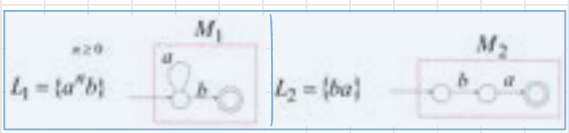
LECTURE 8

Properties of Regular Languages

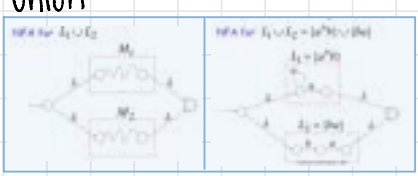
- Union $L_1 \cup L_2$
- Concatenation $L_1 L_2$
- Star L_1^*
- Reversal L_1^R
- Complement $\overline{L_1}$
- Intersection $L_1 \cap L_2$

Any NFA converted to equivalent NFA with single final state

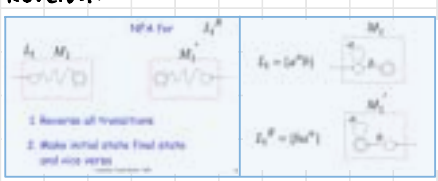
Show regular languages closed under



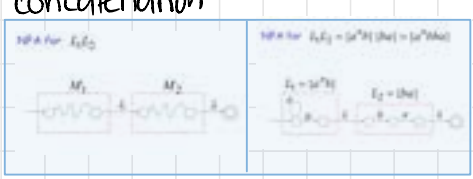
Union



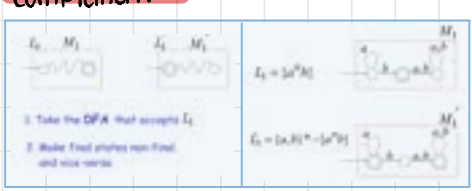
Reversal



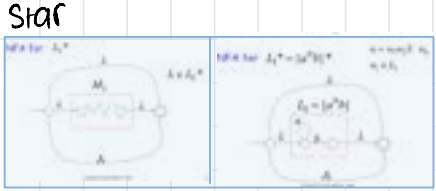
Concatenation



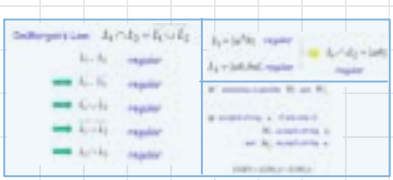
Complement



Star



Intersection



LECTURE 9

MINIMISING DFA's by PARTITION

Why Minimise DFA's

↳ Efficiency

↳ Better understanding of the language

↳ Compare similarity b/w the two FA's

↳ Determine if two DFA's recognise the same language

Minimization Process

↳ Remove inaccesible states

↳ Group equivalent states

↳ 2 states equal if all behaviors same

Minimization Steps

1. Places states of DFA into 2 equivalent classes

↳ final states } 2 partitions

↳ non final states

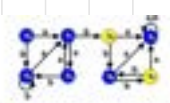
2. Repeat until no more change

↳ for each state in same partition

if transition diff than those in its group

Partition to new equivalence class

8)



- S_0 and S_5 are really the same:
  - Both Final states
  - Both go to S_6 under input b
  - Both go to S_3 under an a
- S_6 and S_7 really the same. Why?
- We say each pair is equivalent

1 final, non final partitions

	a	b
S_0	S_1	S_4
S_1	S_0	S_5
S_3	S_6	S_3
S_4	S_1	S_4
S_5	S_1	S_4
S_6	S_3	S_7
*S_2	S_3	S_6
*S_7	S_3	S_6

S_1, S_6 born end on final states

	a	b
S_0	S_1 I	S_4 I
S_1	S_0 I	S_5 II
S_3	S_6 I	S_3 I
S_4	S_1 I	S_4 I
S_5	S_1 I	S_4 I
S_6	S_3 I	S_7 II
*S_2	S_3 I	S_6 I
*S_7	S_3 I	S_6 I

S_0, S_4, S_5 same partition unlike S_1

	a	b
S_0	S_1 III	S_4 I
S_3	S_6 I	S_3 I
S_4	S_1 III	S_4 I
S_5	S_1 III	S_4 I
S_1	S_0 I	S_5 II
S_6	S_3 I	S_7 II
*S_2	S_3 I	S_6 III
*S_7	S_3 I	S_6 III

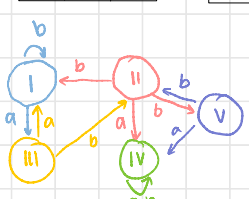
S_1, S_6 not same partion for a

	a	b
S_0	S_1 III	S_4 I
S_4	S_1 III	S_4 I
S_5	S_1 III	S_4 I
S_3	S_3 IV	S_3 IV
S_1	S_0 I	S_5 II
S_6	S_3 IV	S_7 II
*S_2	S_3 IV	S_6 III
*S_7	S_3 IV	S_6 III

	a	b
S_0	S_1 III	S_4 I
S_4	S_1 III	S_4 I
S_5	S_1 III	S_4 I
S_3	S_3 IV	S_3 IV
S_1	S_0 I	S_5 II
S_6	S_3 IV	S_7 II
*S_2	S_3 IV	S_6 V
*S_7	S_3 IV	S_6 V

	a	b
I	III	I
II	IV	V
III	I	II
IV	IV	IV
V	IV	II

5 partitions in order

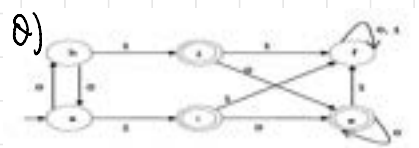


MINIMISING DFA's by Myphill-Nerode Theorem
TF ALGORITHM

Minimization steps

1. Draw table for all pairs of state P,Q
2. Mark every pair in DFA where P → final, Q → not final, vice versa
   ↳ doesn't have to be connected
3. Repeat until unable to mark more
   ↳ check for unmarked pairs $\delta(Q,x)$, $\delta(P,x)$
   UM ↳ unmarked pair → leave it as is
   DE ↳ pair doesn't exist → leave it as is
   M ↳ marked pair → mark Q,P
4. Combine unmarked pairs and make single state

Q)



Step 1, 2

	a	b	c	d	e	f
b						
c	✓	✓				
d	✓	✓				
e	✓	✓				
f	✓	✓	✓	✓	✓	

Step 3

unmarked pairs

(a,b) → $\delta(a,0)=b$ UM, $\delta(a,1)=c$ UM
$\delta(b,0)=a$ $\delta(b,1)=d$

(c,d) → $\delta(c,0)=e$ DE, $\delta(c,1)=f$ DE
$\delta(d,0)=e$ $\delta(d,1)=f$

(e,c) → $\delta(e,0)=e$ DE, $\delta(e,1)=f$ DE
$\delta(c,0)=e$ $\delta(c,1)=f$

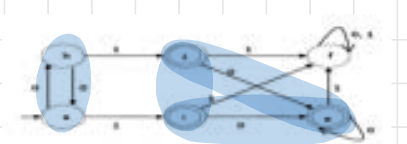
(e,d) → $\delta(e,0)=e$ DE, $\delta(e,1)=f$ DE
$\delta(d,0)=e$ $\delta(d,1)=f$

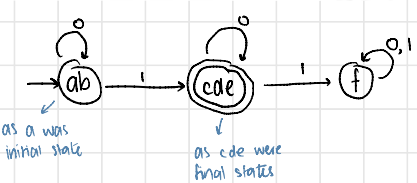
(f,a) → $\delta(f,0)=f$ UM, $\delta(f,1)=f$ M
$\delta(a,0)=b$ $\delta(a,1)=c$ M
so mark f,a

(f,b) → $\delta(f,0)=f$
$\delta(b,0)=a$ M

Step 4

(a,b) (c,d) (e,c) (e,d)





PUMPING LEMMA

used to prove that a language is not regular by contradiction

Pumping Lemma steps

1. Assume A is a regular language → for contradiction

   $|s| \geq P$ → assume

   Divide S into x y z

   $S = xyz$

6. Show 3 Pumping conditions unsatisfied

   ↳ 1. $xy^iz \in A$ for every $i \geq 0$

   ↳ 2. $|y| > 0$

   ↳ 3. $|xy| \leq P$

7. S cannot be pumped = CONTRADICTION

CAN NOT BE USED TO PROVE A LANGUAGE IS REGULAR

Q) Prove $A = \{a^n b^n \mid n \geq 0\}$ is not regular

↳ Assume A is regular → for contradiction

Pumping length = P

$S = a^P b^P$ taking P=7

x y z

aaaaaaa bbbbbbb

case 1: y is in the 'a' part

aaaaaaa bbbbbbb

x y z

$xy^iz \Rightarrow xy^2z$

aa aaaaaaaa abbbbbbb

repeat 2 for y^2

a = b → as $a^n b^n$

1. $11 \neq 7$ ✗
2. $|y| = 4$ ✓
3. $|xy| = 6$ P=7 ✓

case 2: y is in the 'b' part

aaaaaaa bbbbbbb

x y z

$xy^2z \Rightarrow xy^2z$

aaaaaaa bb bbbbbbbb b

y^2

a = b

3. $7 \neq 11$ ✗
2. $|y| = 4$ ✓
3. $|xy| = 13$ P=7 ✗

case 3: y is in the 'a' and 'b' part

aaaaaaabbbbbbb

x y z

$xy^iz \Rightarrow xy^2z$

aaaaa aabbaabb bbbbb

y^2

pattern should be as followed by bs

1. pattern not followed ✗
2. $|y| = 4$ ✓
3. $|xy| = 9$ P=7 ✗

HENCE NOT REGULAR

8) prove $A=\{yy \mid y \in \{0,1\}^*\}$ is not regular

Assume A is regular

Pumping length = P

$S = 0^P 1 0^P 1$ → choose any string

taking P = 7 → choose any

case 1: y is in the '0' part

y^2

00 0000 0000 01000000001

1. doesn't follow pattern yy ✗
   ↳ start and end pattern should be the same
2. $|y| = 4$ ✓
3. $|xy| = 6$ $P = 7$ ✓

Hence not regular

DFA WITH OUTPUTS

output depends on input / doesn't depends on input

Mealy State Machines

↳ Output depends on current state and input

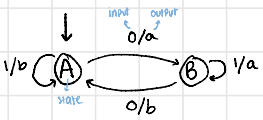
↳ inputs = outputs

↳ no final state

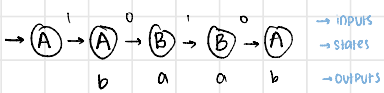
NO FINAL STATE

$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

- Q: finite set of states
- Σ: inputs
- Δ: output
- δ: transition function $Q \times \Sigma \to Q$
- λ: output function $\Sigma \times Q \to \Delta$
- q_0: initial state



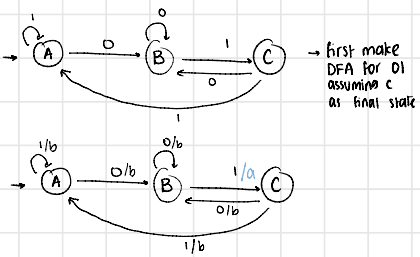
eg 1010



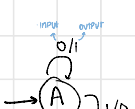
Q) construct mealy machine that produces a when 01 is input string

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

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



Q) construct mealy machine that produces 1's complement



10100
01011

↳ aka complementing machine

Moore State Machines

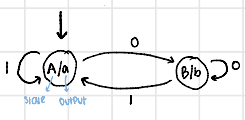
↳ Output depends on current state

↳ inputs = outputs + 1

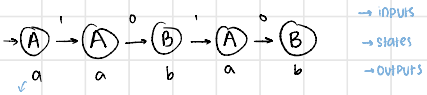
↳ no final state

$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

- Q: finite set of states
- Σ: inputs
- Δ: output
- δ: transition function $Q \times \Sigma \to Q$
- λ: output function $Q \to \Delta$
- q_0: initial state



eg 1010

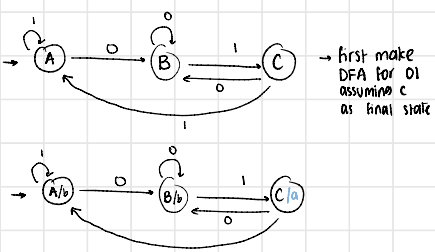


directly comes without any input

Q) construct moore machine that produces a when 01 is input string

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

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



THEOREM

Mealy ⇄ Moore
↳ ignore extra output

Mealy → Moore

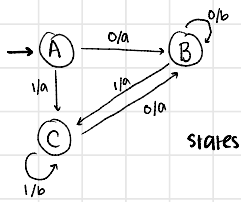
↳ no of states increased — max no of moores state = mealy states × outputs

↳ output of the state = incoming edges output

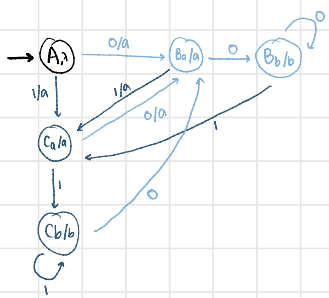
↳ if state already has a diff output

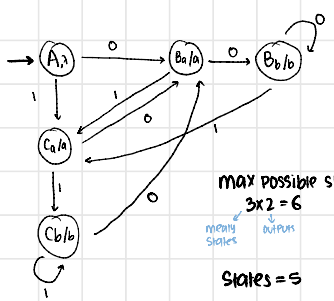
↳ create new state

↳ according to outputs make edges for new states



States = 3





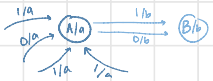
max possible states
3 × 2 = 6
(mealy states × outputs)

States = 5

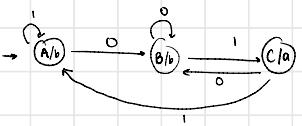
Moore → Mealy

↳ no of states same

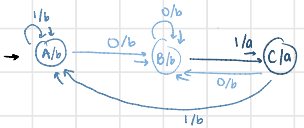
↳ incoming edges output = output of the state



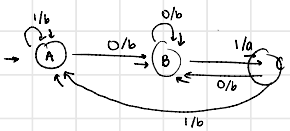
Q)



Old state	0	1	output
→ A	B	A	b
B	B	C	b
C	B	A	a



Old state	0	1	output
→ A	B/b	A/b	b
B	B/b	C/a	b
C	B/b	A/b	a

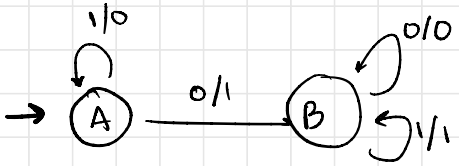


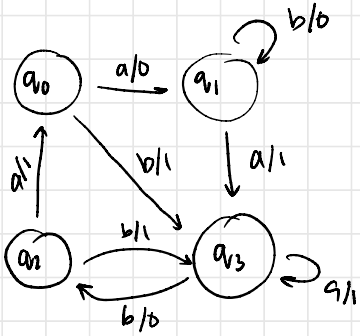
Old state	0	1
→ A	B/b	A/b
B	B/b	C/a
C	B/b	A/b

PRAD
long

mereko

1 0 0 1 1 0 0 1 1 1
1
1 0 0 1 1 0 1 0 0 0





G_1

$S \to abS$

$S \to a$

$L(G_1) = (ab)^*a$

G_2

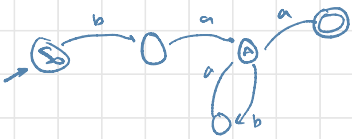
$S \to Aab$

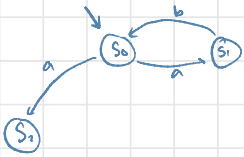
$A \to Aab \mid B$

$B \to a$

$S \to baA$

$A \to baA \mid a$





Grammar

$G = (V, T, S, P)$

- V: Variables
- T: Terminal Symbols
- S: start Symbol
- P: Production rule

Q) $G = (\{S, A, B\}, \{a, b\}, S, \{S \to AB,\ A \to a,\ B \to b\})$

$S \to AB \to aB \to ab$

Q) $G = (\{S\}, \{a, b\}, S, \{S \to aSb \mid \lambda\})$

$S \to aSb \to aaSbb \to aaaSbbb \to aaabbb$

Language of Grammar

Q) $G = \{S \to Ab,\ A \to aAb \mid \lambda\}$

$L(G) = \{a^n b^n b : n \geq 0\}$

Q) $G = \{S \to aSb \mid \lambda\}$

$L = \{a^n b^n,\ n \geq 0\}$

Sentenial form

↳ sentence containing variables and Terminals

Q) $S \to aSb \to aaSbb \to aaaSbbb \to aaabbb$

Sentenial forms: aSb, $aaSbb$, $aaaSbbb$; Sentence: $aaabbb$

* sign

Write	instead of
$S \xrightarrow{*} aabb$	$S \to aSb \to aaSbb \to aabb$
$w_1 \xrightarrow{*} w_n$	$w_1 \to w_2 \to w_3 \to \ldots \to w_n$

Q) $G = \{S \to aSb \mid \lambda\}$

↳ Derivations

$S \xrightarrow{*} \lambda$

$S \xrightarrow{*} aaSbb$

$S \xrightarrow{*} ab$

$S \xrightarrow{*} aabb$

$S \xrightarrow{*} aaabbb$

Q) $G = \{S \to Ab,\ A \to aAb \mid \lambda\}$

↳ Derivations

$S \xrightarrow{*} \lambda$

$S \xrightarrow{*} Ab \xrightarrow{*} b$

$S \xrightarrow{*} Ab \xrightarrow{*} aAbb \xrightarrow{*} abb$

$S \xrightarrow{*} Ab \xrightarrow{*} aAbb \xrightarrow{*} aaAbbb \xrightarrow{*} aabbb$

$S \xrightarrow{*} a^n b^n b$

Linear Grammars

↳ G with 1 or 0 variable in a production

a) $S \to aSb|Ab|\lambda$, $A \to aAb|\lambda$ ✓

b) $S \to SS|aSb|bSa|\lambda$ ✗ $L(G) = \{w : n_a(w) = n_b(w)\}$ (number of a in string w)

c) $S \to A$, $A \to aB|\lambda$, $B \to Ab$ ✓ $L(G) = \{a^n b^n : n \geq 0\}$

Right Linear Grammar

↳ all productions form

- ↳ no variable
- ↳ variable but only in the end

$A \to xB$ or $A \to x$

(x: string of terminals)

a) $S \to abS$

b) $S \to a$

Left Linear Grammar

↳ all productions form

- ↳ no variable
- ↳ variable but only in the start

$A \to Bx$ or $A \to x$

(x: string of terminals)

a) $S \to Aab$

b) $S \to Aab|B$

c) $S \to a$

Regular Grammar

↳ It is a Right Linear / Left Linear Grammar

↳ Regular grammars generate only regular languages

- ↳ Union $L_1 \cup L_2$
- ↳ Concatenation $L_1 L_2$
- ↳ Star L_1^*
- ↳ Reversal L_1^R
- ↳ Complement $\overline{L_1}$
- ↳ Intersection $L_1 \cap L_2$

a) $G = \{S \to abS|a\}$

$L(G) = (ab)^*a$

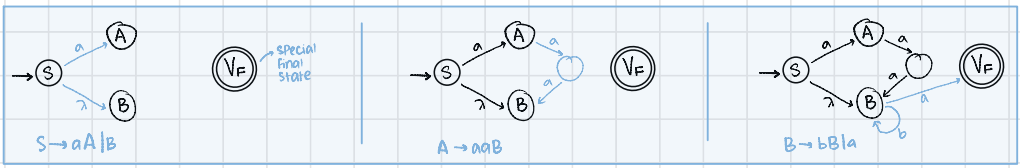
b) $G = \{S \to Aab, A \to Aab|B, B \to a\}$

$L(G) = aab(ab)^*$

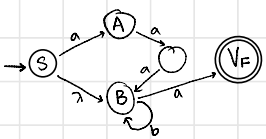
Proof Right Linear Grammar is Regular → part 1

↳ construct NFA M
such that L(M) = L(G)

0) G = { S → aA | B , A → aaB , B → bB | a }



NFA M



$L(M) = L(G) = aaab^*a + b^*a$

Proof Left Linear Grammar is Regular

↳ construct right linear grammar G'
such that $L(G) = L(G)^R$

0) G = { A → BabC_k | Bv } → left linear G
G' = { A → kCbaB | v*B } → right linear G

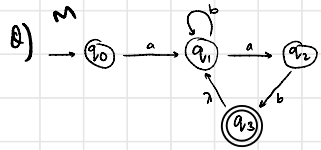
0) G = { A → abC_k | v } → left linear G
G' = { A → kCba | v^R } → right linear G

Proof Regular Language is Linear Grammar → part 2

↳ construct from M a Regular grammar G

↳ such that L(M)=L(G)

a)

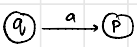


$L = ab^*ab(b^*ab)^*$

$L = L(M)$

convert to Right Linear Grammar

1. For any transition



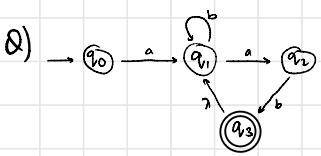
$q \longrightarrow aP$

variable — terminal — variable

2. For any final transition

$q_f \longrightarrow \lambda$

a)



$q_0 \rightarrow aq_1$

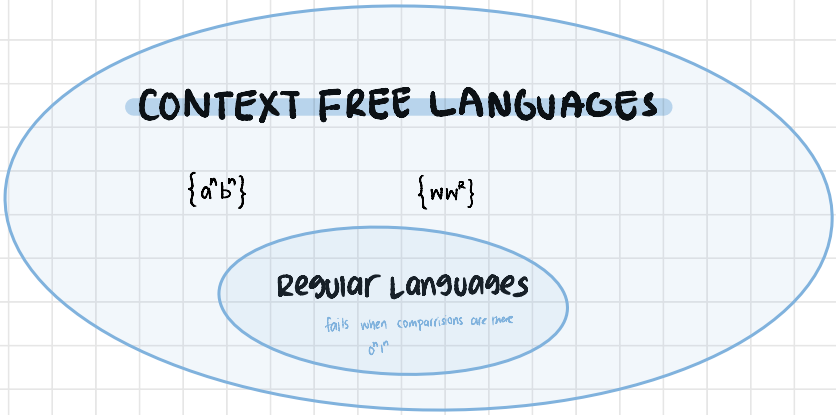
$q_1 \rightarrow bq_1$

$q_1 \rightarrow aq_2$

$q_2 \rightarrow bq_3$

$q_3 \rightarrow q_1$

$q_3 \rightarrow \lambda$



context free languages

context free Grammar

Q) generate equal no of a's and b's in the form $a^n b^n$ (not regular lang)

$G = \{ (S,A), (a,b), (\overbrace{S \to aAb}^{P}, A \to aAb \mid \epsilon) \}$ (ε: epsilon, empty symbol)

$S \to aAb$ (replace $A \to aAb$)

$\to aaAbb$ (replace $A \to aAb$)

$\to aaaAbbb$ (replace $A \to \epsilon$)

$\to aaabbb$

$\to a^3 b^3 \to a^n b^n$

$V = \{S, A\}$

$T = \{a, b\}$

CFL to CFG

1) $a^n b^n$, $n \geq 1$

$S \to aSb \mid \lambda$

2) $a^n b^{n+2}$, $n \geq 0$

$S \to aSb \mid bb$

3) $a^{2n} b^n$

$S \to aaSb \mid \lambda$

4) $a^{2n+3} b^n$, $n \geq 0$

$S \to aaSb \mid aaa$

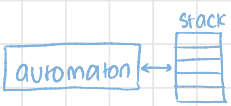
5) $a^m b^n$, $m \geq n$

$S \to AS_1$

$S_1 \to aS_1b \mid \lambda$

$A \to aA \mid a$

Pushdown Automata



6) $\{w \mid n_a(w) = n_b(w)\}$ ($a^n b^n + b^n a^n$)

$S \to aSb \mid bSa \mid SS \mid \lambda$ (SS → abba)

7) $ww^R \cup w(a+b)w^R$

$S \to aSa \mid bSb \mid a \mid b \mid \lambda$

8) $a^m b^m c^n$ $m,n \geq 0$

$S \to S_1C$

$S_1 \to aS_1b \mid \lambda$

$C \to cC \mid \lambda$

Context Free Grammar

$G=(V,T,S,P)$

- variables
- start variable
- Production rule
- Terminal symbols

$A \to a$
$a=\{V \cup \Sigma\}^*$
$A \in V$

1. $G=\{S \to aSb|\lambda\}$ (production rule)

$S \to aSb \to aaSbb \to aabb$

$L(G)=\{a^n b^n : n \geq 0\}$

2. $G=\{S \to asa|bSb|\lambda\}$

multiple derivations:

$S \to aSa \to abSba \to abba$

$S \to aSa \to abSba \to abaSaba \to abaaba$

$L(G)=\{ww^R : w \in \{a,b\}^*\}$

Context free languages

↳ $L=L(G)$

1. $L=\{a^n b^n : n \geq 0\}$, $G=\{S \to aSb|\lambda\}$ ✓ $L(G)=L$

NOTATION : Context Free Grammar

beg lower case letters: a,b,c ...	Terminal Symbols	T
end lower case letters: w,x,y ...	strings of terminals	
beg upper case letters: A,B,C ...	variables	V
end upper case letters: X,Y,...	Terminals/strings	
lower case Greek letters: α,β,γ ...	strings of Terminals/variables	

Derivation order

1. $S \to AB$ 2. $A \to aaA|\lambda$ 3. $B \to Bb|\lambda$

Derrive a string aab

↳ Leftmost derivation order of string

$S \xrightarrow{1} AB \xrightarrow{2} aaAB \xrightarrow{3} aaB \xrightarrow{4} aaBb \xrightarrow{5} aab$

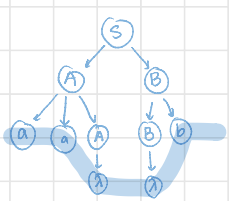
↳ Rightmost derivation order of string

$S \xrightarrow{1} AB \xrightarrow{4} ABb \xrightarrow{3} Ab \xrightarrow{2} aaAb \xrightarrow{3} aab$

Derivation Trees/Parse Tree

$S \to AB$ $A \to aaA|\lambda$ $B \to Bb|\lambda$

$S \to AB \to aaAB \to aaABb \to aab$



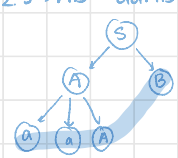
$=aab$

Partial Derivation Tree

$S \to AB$ $A \to aaA|\lambda$ $B \to Bb|\lambda$

1. $S \to AB$

2. $S \to AB \to aaAB$



sentential form

aaAB

Ambiguity → bad for programming

↳ when 2 or more derivation trees

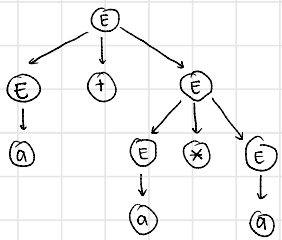
↳ two left most derivation trees

Q) $E \rightarrow E+E \mid E*E \mid E \mid a$

$a+a*a$

$2+2\times2$

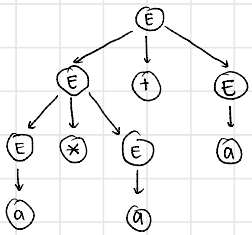
left most derivation



$2+2*2=6$ → CORRECT

$E \rightarrow E+E \rightarrow a+E \rightarrow a+E*E \rightarrow a+a*E \rightarrow a+a*a$

left most derivation



$2+2*2=8$

$E \rightarrow E*E \rightarrow E+E*E \rightarrow a+E*E \rightarrow a+a*E \rightarrow a+a*a$

2 left most derivation tress hence ambigious

How to remove Ambiguity

↳ it is difficult to achive and sometimes possible

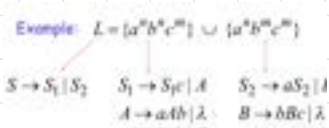
Q) $E \rightarrow E+E \mid E*E \mid E \mid a$ → Ambigious Grammar

CONVERT

$E \rightarrow E+T \mid T$, $T \rightarrow T*F \mid F$, $F \rightarrow E \mid a$ → non Ambigious Grammar

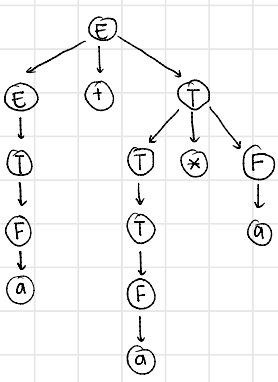
Inherent Ambiguity

↳ will always have ambigious grammar



Unique derivation tree

$a+a*a$



Substitution Rules

1. NULLABE VARIABLES

Ex) $S \to aMb$
$M \to aMb$
$M \to \lambda$
↓ Nullable variable

Substitute $M \to \lambda$

$S \to aMb \mid ab$
$M \to aMb \mid ab$

Ex) $S \to ABC$
$A \to aA \mid \lambda$
$B \to bB \mid \lambda$
$C \to C$

Nullable variable

$S \to ABC \mid AC \mid BC \mid C$
$A \to aA \mid a$
$B \to bB \mid b$
$C \to C$

Ex) $S \to aS \mid A$ (as $A \to \lambda$)
$A \to \lambda$

Nullable variable

Substitute $A \to \lambda$

$S \to aS \mid \lambda$

Substitute $S \to \lambda$

$S \to aS \mid a \mid \lambda$

still here?

* if $L(G) = \lambda$ then dont remove it

2. UNIT Productions

↳ remove single variable on both sides $A \to B$
↳ remove same variable on both sides $A \to A$
↳ remove repeated productions $A \to aB \mid aB$
↳ transitive property $A \to B, B \to a \implies A \to a$

Ex) $S \to aA$
$A \to a$
$A \to B$
$B \to A$
$B \to bb$
↓ unit productions

Substitute $A \to B$

$S \to aA \mid aB$
$A \to a$
$B \to A \mid B$
$B \to bb$

Remove $B \to B$

$S \to aA \mid aB$
$A \to a$
$B \to A$
$B \to bb$

Substitute $B \to A$

repeated

$S \to aA \mid aB \mid aA$
$A \to a$
$B \to bb$

$S \to aA \mid aB$
$A \to a$
$B \to bb$

Ex) $S \to AB$
$A \to a$
$B \to C \mid b$
$C \to D$
$D \to E$
$E \to a$
↓ unit productions

transitive Property

$S \to AB$
$A \to a$
$B \to a \mid b$
$C \to a$
$D \to a$
$E \to a$

unreachable transitions

Remove C, D, E

$S \to AB$
$A \to a$
$B \to a \mid b$

3. USELESS PRODUCTIONS

↳ production that cause derivations to never end

Q) $S \to aSb$
$S \to \lambda$
$S \to A$
$A \to aA$ ← useless production

Q) $S \to aSb$
$S \to \lambda$
$S \to A$
$A \to aA$
$B \to C$
$C \to D$
→ USELESS

TO REMOVE USELESS PRODUCTIONS

↳ find all variable that can produce strings with only terminals

↳ keep only variables that use other variables

↳ Find all rechable variables from S

Q) $S \to aS|A|C$
$A \to a$
$B \to aa$
$C \to aCb$

→ ONLY TERMINALS ROUND 1 $\{A, B, S\}$

$S \to aS|A$
$A \to a$
$B \to aa$

→ variables rechable from S

$S \to aS|A$
$A \to a$

Simplication of CFG

1. remove nullable variables
2. remove unit productions
3. remove useless productions

Q) $S \to aB$
$A \to aaA$
$A \to abBc$
$B \to aA$
$B \to b$

Substitute $B \to b$

$S \to aB|ab$
$A \to aaA$
$A \to abBc|abbc$
$B \to aA$

Substitute $B \to aA$

$S \to aB|ab|aaA$
$A \to aaA$
$A \to abBc|abbc|abaAc$

$S \to ab|aaA$
$A \to aaA$
$A \to abbc|abaAc$

→ Equivalent Grammar
use substitution rules

NORMAL FORMS FOR CONTEXT FREE GRAMMAR

Chomsky Normal Form

↳ Terminal $A \to a$

↳ Variable Variable $A \to BC$

1. $S \to AS \mid a$, $A \to SA \mid b$ ✓
2. $S \to AS \mid AAS$, $A \to SA \mid aa$ ✗

Greinbach Normal Form

↳ Terminal 1 or 0 $S \to a$

↳ Terminal, infinite variables $S \to aABCD$

↳ infinite variables $S \to ABC$

1. $S \to cAB$, $A \to aA \mid bB \mid b$, $B \to b$ ✓
2. $S \to abSb \mid aa$ ✗

Convert to Chomsky Normal Form

↳ shouldn't contain λ → else do substitution

↳ add new variables for terminals

0) $S \to ABa$, $A \to aab$, $B \to Ac$

Introducing I, J, K:

$S \to ABI$, $A \to IIJ$, $B \to AK$, $I \to a$, $J \to b$, $K \to c$

Introducing X, Y:

$S \to AX$, $X \to BI$, $A \to IY$, $Y \to IJ$, $B \to AK$, $I \to a$, $J \to b$, $K \to c$

$S \to AX$, $X \to BI$, $A \to IY$, $Y \to IJ$, $B \to AK$, $I \to a$, $J \to b$, $K \to c$ → Chomsky

Convert to Greinbach Normal Form

↳ Change Terminal symbols to variables

0) $S \to abSb$, $S \to aa$

Introducing I, J:

$S \to IbSJ$, $S \to aI$, $I \to a$, $J \to b$

$S \to IbSJ$, $S \to aI$, $I \to a$, $J \to b$ → Greinbach

CYK Algorithm → only applicable for chomsky normal form

↳ checks wether string belongs to a CFG or not

1. convert to CNF if not
2. fill 1st diagonals with ther derivatives
3. fill 2nd diagonals with concatenated derivatives

⋮

CHOMSKY NORMAL FORM

$A \to \overset{V}{B}\overset{V}{C}$ $A \to \overset{T}{a}$

Q) $S \to AB$
$A \to BB|a$
$B \to AB|b$

abbb valid member or not

1	2	3	4
a	b	b	b
A	B	B	B

→ diagonal 1

AB BB BB → diagonal 2
SB A A

if S in (1,4) then valid
if no S in (1,4) then invalid

	4	3	2	1
1	SB	A	S,B	A
2	S,B	A	B	
3	A	B		
4	B			

valid

1,3 → diagonal 3 ← 2,4

1|2|3 2|3|4

11,23 AA × 12,33 SB B SB BB × A

22,34 BA × 23,44 AB SB

1,4 → diagonal 4

1|2|3|4

11,24 A SB As × 12,34 SB A AB SB × 13,44 A B SB SA BA × ×

Pumping Lemma for CFL

1. Assume A is a regular language → for contradiction

   $|s| \geq P$ → assume
   Divide S into 5 Pieces

   $S = uvxyz$

2. Show 3 pumping conditions unsatisfied

   1. $uv^i x y^i z$, $i \geq 0$
   2. $|vy| > 0$
   3. $|vxy| \leq P$

3. S cannot be pumped = CONTRADICTION

---

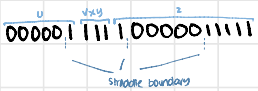
8) $L = \{ww \mid w \in \{0,1\}^*\}$ is NOT context free · SHOW

assume L is context free

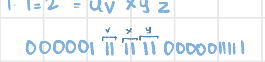
$S \geq 5$ assume

$S = \underbrace{01}_{w}\underbrace{01}_{w}$

Case 1: vxy no straddle boundary

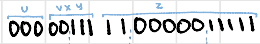


1. $i=2$ (assume) $= uv^2xy^2z$ ✗



$0^5 0 1^5$ first half ≠ second half as wrong pattern

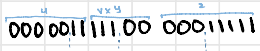
Case 2a: vxy straddle first boundary



1. $i=2 = uv^2xy^2z$ ✗

$0^3 1^3 0^5 1^5$ first half ≠ second half as wrong pattern

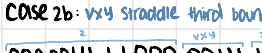
Case 3 : vxy straddle the midpoint



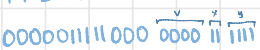
1. $i=2 = uv^2xy^2z$ ✗

$0^5 1^3 0^3 1^5$ first half ≠ second half as wrong pattern

Case 2b: vxy straddle third boundary



1. $i=2 = uv^2xy^2z$ ✗



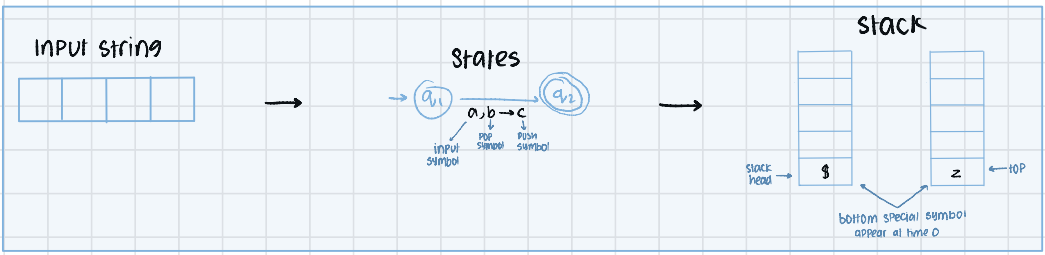
$0^5 1^5 0^3 1^3$ first half ≠ second half as wrong pattern

HENCE NOT CONTEXT FREE

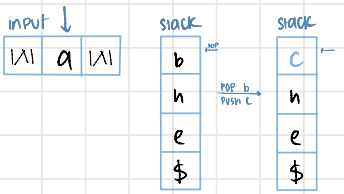
Pushdown Automata (PDA)

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

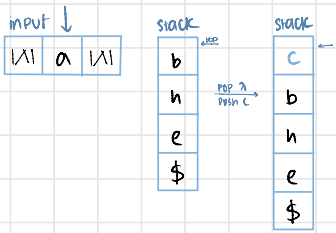
- Q: states
- Σ: input alphabets
- Γ: stack
- δ: transition function
- q_0: initial state
- z: stack start symbol
- F: accepted states



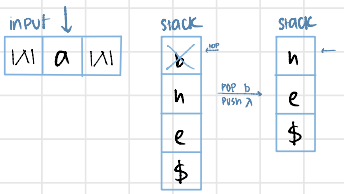
1. $(q_1) \xrightarrow{a,b \to c} (q_2)$



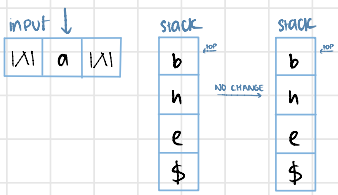
2. $(q_1) \xrightarrow{a,\lambda \to c} (q_2)$



3. $(q_1) \xrightarrow{a,b \to \lambda} (q_2)$



4. $(q_1) \xrightarrow{a,\lambda \to \lambda} (q_2)$



Instantaneous Description

(q, u, s)

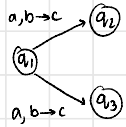
- q: current state
- u: remaining input
- s: current stack contents

* stack contents dont matter

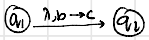
Non Deterministic PDAs

↳ PDAs are non deterministic

↳ allowed non deterministic transitions



λ - transition



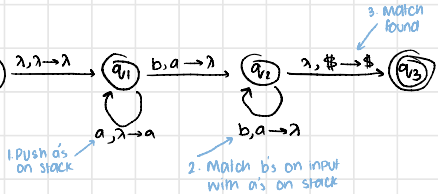
↳ A string is accepted if

↳ all the input is consumed

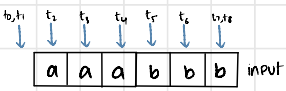
↳ the last state is an accepting state

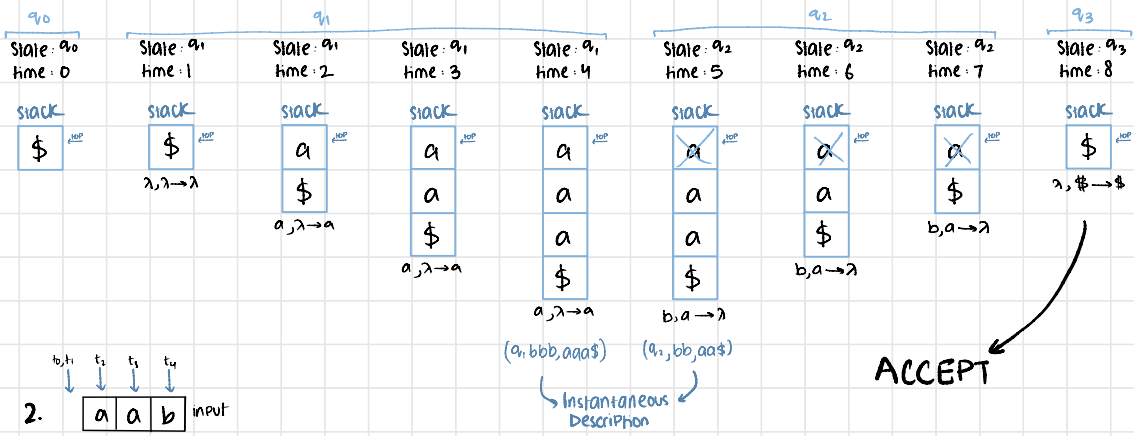
} both conditions to be fulfilled

Q) PDA M : $L(M) = \{a^n b^n : n \geq 0\}$



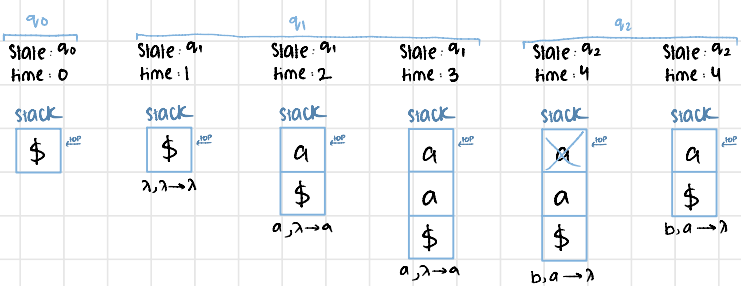
1.





ACCEPT

2.



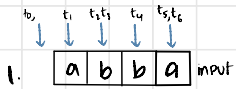
↳ REJECT

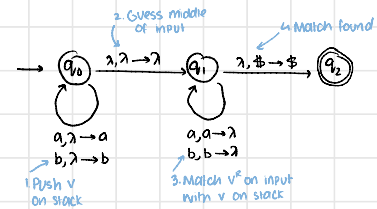
as no accepted state reached and inputs all consumed

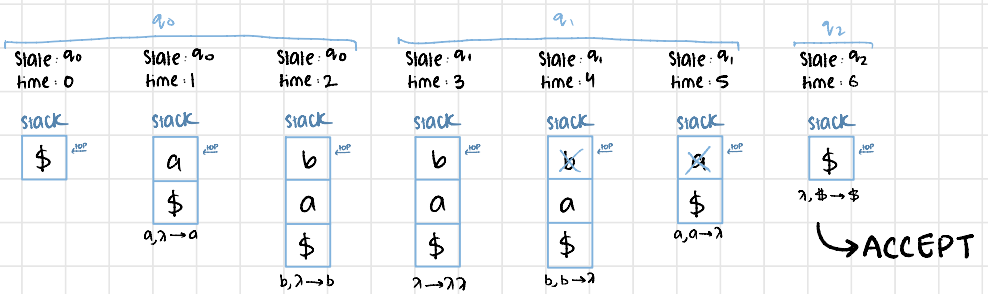
hence no accepting computation for aab

Q) PDA M: $L(M) = \{ vv^R : v \in \{a,b\}^* \}$ → Palindrome

1.

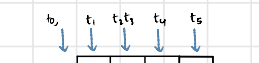


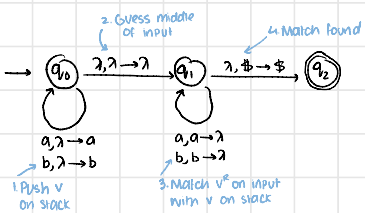


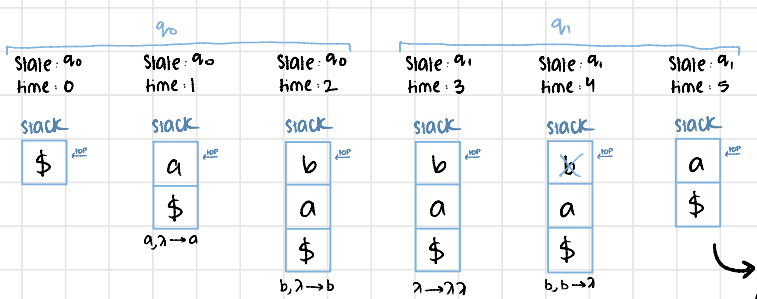


↳ ACCEPT

2.

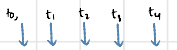


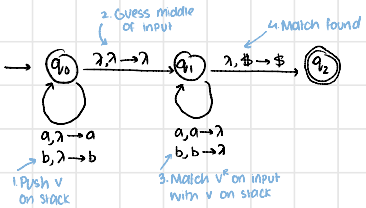




↳ REJECT
as can't consume input

3.





State: q_0 Time: 0 — Stack: $ (top)

State: q_0 Time: 1 — Stack: a (top), $ — $a,\lambda \to a$

State: q_0 Time: 2 — Stack: b (top), a, $ — $b,\lambda \to b$

State: q_0 Time: 3 — Stack: b (top), b, a, $ — $b,\lambda \to b$

State: q_0 Time: 4 — Stack: b, b, b (top), a, $ — $b,\lambda \to b$

State: q_1 Time: 5 — Stack: b, b, b (top), a, $ — $\lambda \to \lambda\lambda$

↳ REJECT

as no accepted state reached and inputs all consumed

A computation:

$(q_0, aaabbb, \$) \vdash (q_1, aaabbb, \$) \vdash$

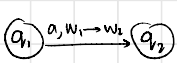
$(q_1, aabbb, a\$) \vdash (q_1, abbb, aa\$) \vdash (q_1, bbb, aaa\$) \vdash$

$(q_2, bb, aa\$) \vdash (q_2, b, a\$) \vdash (q_2, \lambda, \$) \vdash (q_3, \lambda, \$)$

For convenience we write:

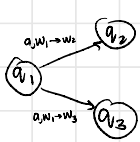
$(q_0, aaabbb, \$) \overset{*}{\vdash} (q_3, \lambda, \$)$

Formalities for PDA's



Transition function

$\delta(q_1, a, w_1) = \{(q_2, w_2)\}$



Transition function

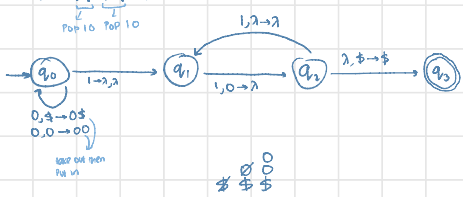
$\delta(q_1, a, w_1) = \{(q_2, w_2), (q_3, w_3)\}$

Construct a PDA

Q) $L_1 = \{0^n 1^{2n}, n>0\}$

00 | | | | λ

Pop 10 Pop 10



Transition states

$q_0, 0, \$ \to q_0, 0\$$

$q_0, 0, 0 \to q_0, 00$

$q_0, 1, \lambda \to q_1, \lambda$

$q_1, 1, 0 \to q_2, \lambda$

$q_2, 1, \lambda \to q_1, \lambda$

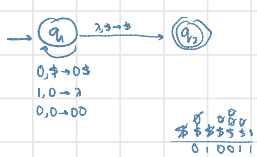
$q_2, \lambda, \$ \to q_3, \$$

Q) $L_1 \{ w \in (0,1)^* \mid n_0(w) = n_1(w)$

↳ no of 0,1s same

01, 10, 0011, 010011

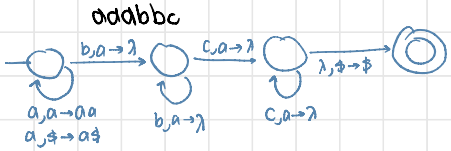
010011



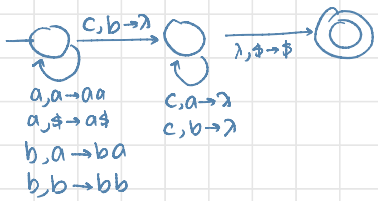
what about 1,$→1$

Q) $L_1 = \{a^{m+n} b^m c^n\}$

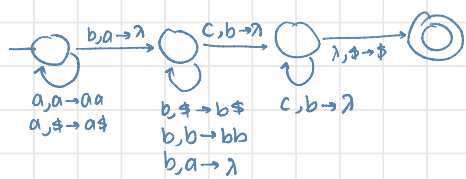
aaabbc



Q) $L_1 = \{a^n b^m c^{n+m}\}$

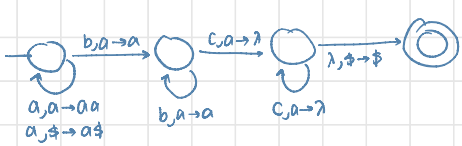


Q) $L_1 = \{a^n b^{m+m} c^m\}$

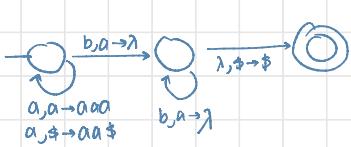


Q) $L_1 = \{a^n b^m c^n\}$ $n,m \geq 1$

aba abba aabaa ✓



Q) $L_1 = \{a^n b^{2n}\}$



Language of PDA

$L(M) = \{w : (q_0, w, z) \overset{*}{} (q_f, \lambda, s)\}$

- q_0: initial state
- q_f: accepted state

Convert CFG to PDAs

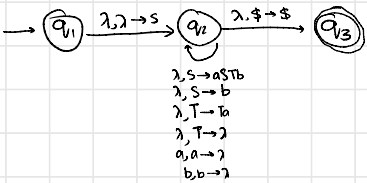
↳ $L(G) = L(M)$

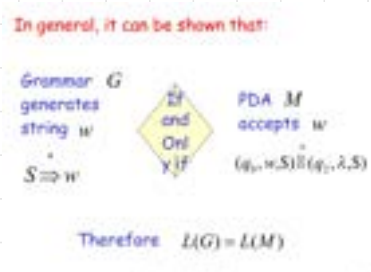
↳ for each production, add transitions $A \to w \Rightarrow \lambda, A \to w$

↳ for each terminal, add transitions $a \Rightarrow a, a \to \lambda$

1.

Grammar	PDA	
$S \to aSTb$	$\lambda, S \to aSTb$	
$S \to b$	$\lambda, S \to b$	
$T \to Ta$	$\lambda, T \to Ta$	$a, a \to \lambda$
$T \to \lambda$	$\lambda, T \to \lambda$	$b, b \to \lambda$





So far we have shown:

$L(G) \subseteq L(M)$

With a similar proof we can show

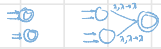
$L(G) \supseteq L(M)$

Therefore: $L(G) = L(M)$

convert PDAs to CFG

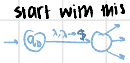
↳ $L(M) = L(G)$

1. 1 final state



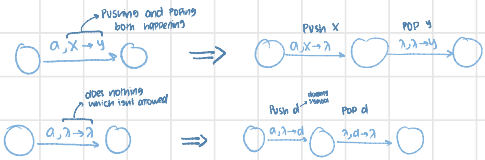
2. start and end with empty stack

start with this



end with this

3. assure each transition Pushes/Pops, but not both



4. NO underflow or overflow stack

↳ Popping on empty stack

↳ Pushing when stack is full

Positive closure for CFL

↳ context free

↳ has new start variable S

↳ has new additional production $S \to S_1 | S_2$ / $S \to S_1 S_2$ / ...

↳ UNION → CONTEXT FREE

$L_1 = \{a^n b^n\}$ $\quad S_1 \to aS_1b | \lambda$

$L_2 = \{ww^R\}$ $\quad S_2 \to aS_2 | bS_2b | \lambda$

$L_1 \cup L_2 = \{a^n b^n\} \cup \{ww^R\}$ $\quad S \to S_1 | S_2$

↳ CONCATENATION → CONTEXT FREE

$L_1 = \{a^n b^n\}$ $\quad S_1 \to aS_1b | \lambda$

$L_2 = \{ww^R\}$ $\quad S_2 \to aS_2 | bS_2b | \lambda$

$L_1 L_2 = \{a^n b^n\}\{ww^R\}$ $\quad S \to S_1 S_2$

↳ STAR OPERATION → CONTEXT FREE

$L = \{a^n b^n\}$ $\quad S \to aSb | \lambda$

$L = \{a^n b^n\}^*$ $\quad S_1 \to SS_1 | \lambda$

Negative closure for CFL

↳ not necessarily context free

↳ INTERSECTION → NOT NECESSARILY CONTEXT FREE

$L_1 = \{a^n b^n c^m\}$ $\quad S \to AC$, $A \to aAb | \lambda$, $C \to cC | \lambda$

$L_2 = \{a^n b^m c^m\}$ $\quad S \to AB$, $A \to aA | \lambda$, $B \to bBc | \lambda$

$L_1 \cap L_2 = \{a^n b^n c^n\}$ NOT context free

↳ COMPLEMENT → NOT NECESSARILY CONTEXT FREE

$L_1 = \{a^n b^n c^m\}$ $\quad S \to AC$, $A \to aAb | \lambda$, $C \to cC | \lambda$

$L_2 = \{a^n b^m c^m\}$ $\quad S \to AB$, $A \to aA | \lambda$, $B \to bBc | \lambda$

$\overline{L_1 \cup L_2} = L_1 \cap L_2 = \{a^n b^n c^n\}$ NOT context free

Intersection of Context Free Language and Regular Language

L_1 context free
L_2 regular
} $L_1 \cap L_2$ → context free

Machine M_1
PDA for L_1
context free

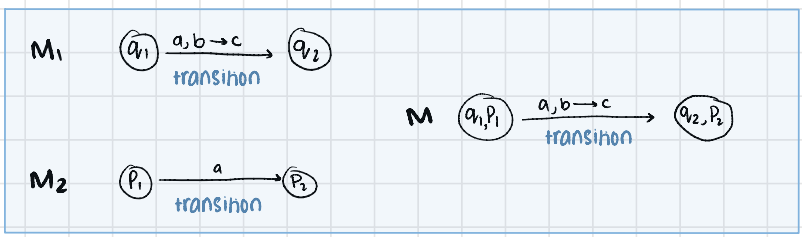
Machine M_2
DFA for L_2
regular

M simulates in parallel M_1 and M_2

M accepts string w if and only if

M_1 accepts string w and
M_2 accepts string w

$$L(M) = L(M_1) \cap L(M_2)$$



Therefore:

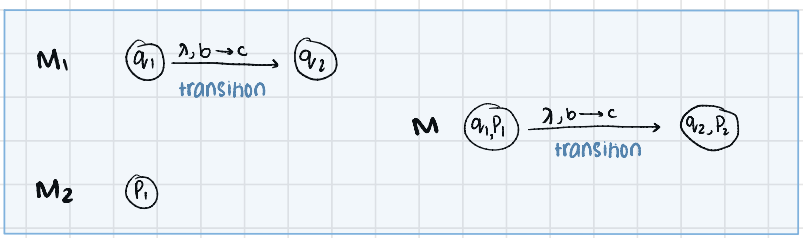
M is PDA

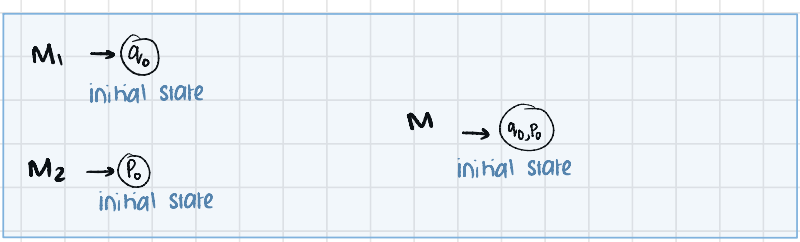
⬇

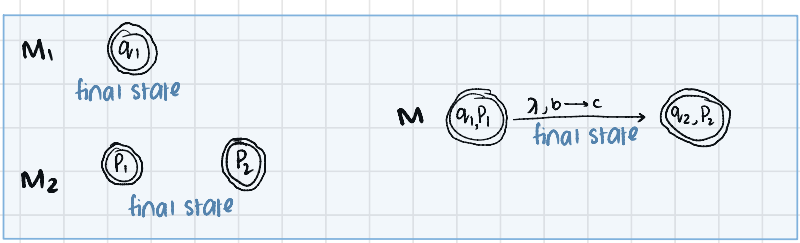
$L(M_1) \cap L(M_2)$ is context-free

⬇

$L_1 \cap L_2$ is context-free







Deterministic PDA (DPDA)

↳ λ must exist

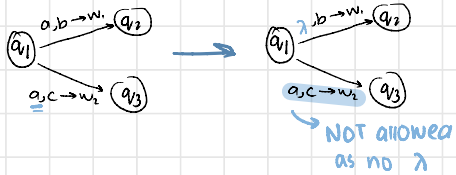
↳ λ, λ → λ ✗

1. 

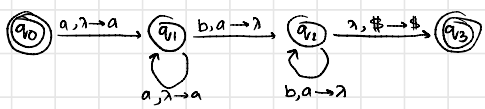
✓

2. 

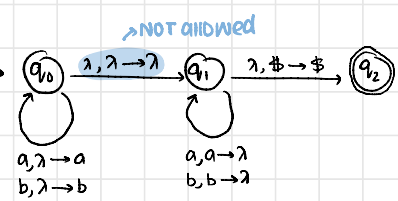
✓

3. 

✗

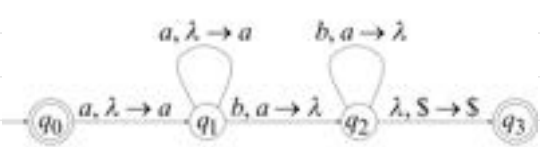
4. 

✓

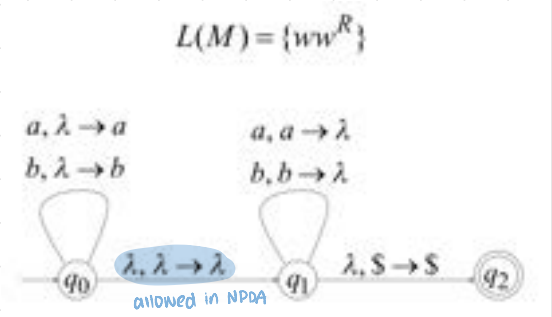
5. 

✗

$$L(M) = \{a^n b^n : n \geq 0\}$$



NON Deterministic PDA (NPDA)



NPDAs are more powerfull than DPDAs

CHECK SLIDES

Proof L is non deterministic CFL
there is no DPDA that accepts L

↳ Proof by contradiction

$a^n b^n c^n$ → PDA fails as more than 2 comparisons

Turing Machines (TM)

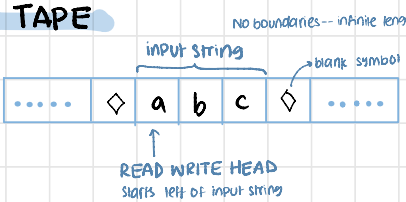
$M = (Q, \Sigma, \Gamma, \delta, q_0, \Diamond, F)$

- Q — states
- Σ — input alphabets
- Γ — TAPE alphabet
- δ — transition function
- q_0 — initial state
- $\Diamond$ — blank
- F — final states

Standard Turing Machine

- Deterministic
- Infinite Tape in both directions
- Tape is the in the input/output file

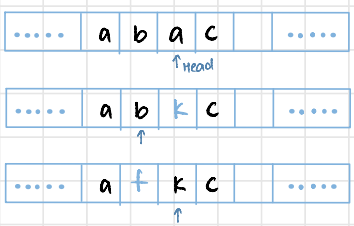
TAPE



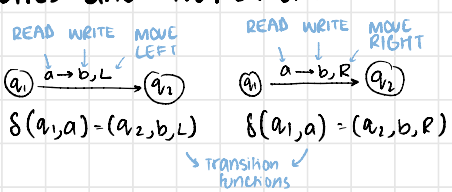
head at each transition
1. Reads a symbol
2. Writes a symbol
3. Moves left/right

CONTROL UNIT

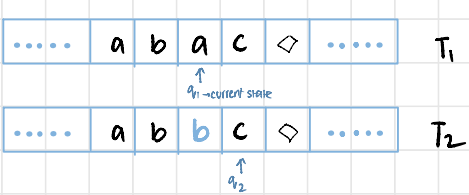
1.



States and transitions



1. q_1 —a → b, L→ q_2

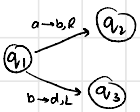


2. q_1 —◇ → g, R→ q_2

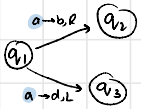
Turing Machines are deterministic

↳ NO same read symbols

ALLOWED



NOT ALLOWED



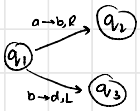
Halting

↳ when no possible transitions

1.

.....	a	b	a	c	◇	.....

$\uparrow q_1$



no possible transition

HALT

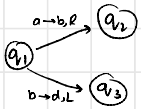
Partial Transition function

↳ NO transition for an input symbol

1.

.....	a	b	a	c	◇	.....

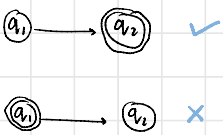
$\uparrow q_1$



No transition for input symbol c

Final states

↳ NO transitions from final state



✓

×

Accept input

↳ if machine halts in final state

we call it

↳ Turing Recognizable

↳ Turing Acceptable

↳ Recursively Enumerable

Reject input

↳ if machine halts in nonfinal state

OR

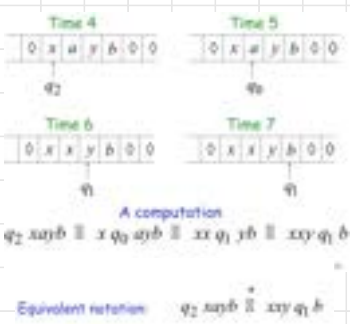
↳ if machine enters infinite loop

Instantaneous Description

.....	c	a	b	a	◇	.....

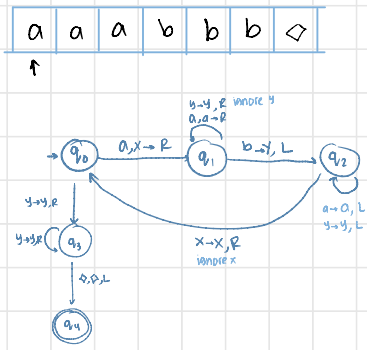
$\uparrow q_1$

ID = ca q_1 ba

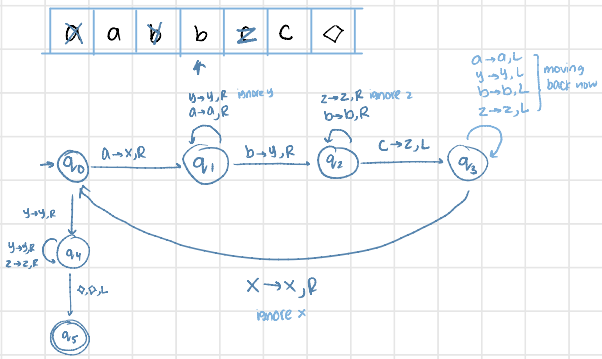


Design Turing Machine

Q) $\{a^n b^n \mid n \geq 1\}$

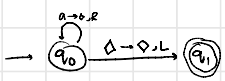


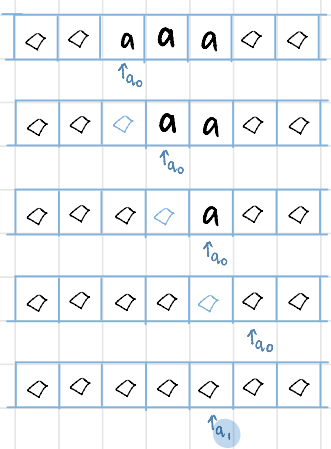
Q) $\{a^n b^n c^n \mid n \geq 1\}$



Turing Machine Examples

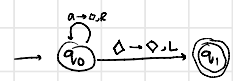
1. A turing machine that accepts the language a^*





Halt and Accept

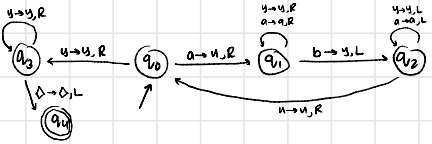
2. A turing machine that accepts the language a^*



NO possible transition

Halt and Reject

3. A turing machine for language $\{a^n b^n\}$



Basic Idea:

Match a's with b's:
Repeat:
replace leftmost a with x
find leftmost b and replace it with y
Until there are no more a's or b's

If there is a remaining a or b reject

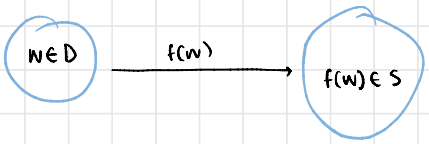
◇	a	a	b	b	◇	◇	T_0 a→u,R
	$\uparrow a_0$						
◇	u	a	b	b	◇	◇	T_1 a→a,R
		$\uparrow a_1$					
◇	u	a	b	b	◇	◇	T_2 b→y,L
			$\uparrow a_1$				
◇	u	a	y	b	◇	◇	T_3 a→a,L
		$\uparrow a_2$					
◇	u	a	y	b	◇	◇	T_4 u→u,R
	$\uparrow a_2$						
◇	u	a	y	b	◇	◇	T_5 a→u,R
		$\uparrow a_0$					
◇	u	u	y	b	◇	◇	T_6 y→y,R
			$\uparrow a_1$				
◇	u	u	y	b	◇	◇	T_7 b→y,L
				$\uparrow a_1$			
◇	u	u	y	y	◇	◇	T_8 y→y,L
			$\uparrow a_2$				
◇	u	u	y	y	◇	◇	T_9 u→u,R
		$\uparrow a_2$					
◇	u	u	y	y	◇	◇	T_{10} y→y,R
			$\uparrow a_0$				
◇	u	u	y	y	◇	◇	T_{11} y→y,R
				$\uparrow a_3$			
◇	u	u	y	y	◇	◇	T_{12} ◇→◇,L
					$\uparrow a_3$		
◇	u	u	y	y	◇	◇	T_{13}
				$\uparrow a_4$			

Halt and Accept

Computing Functions with Turing Machines

A function $f(w)$ has

Domain D — Result Region S



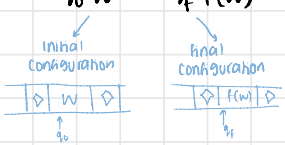
Decimal: 5

Binary: 101

Unary: 11111 → preferd as easier to manipulate with Turing machine

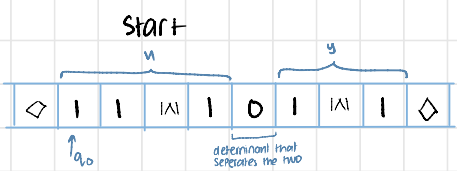
↳ A function f is computable if there is a Turing Machine M such that

$q_0\, w \;*\; q_f\, f(w)$



1. $f(n,y) = n + y$

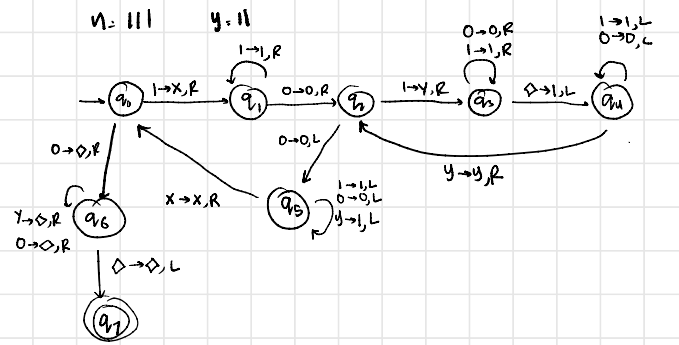
input string: n 0 y unary

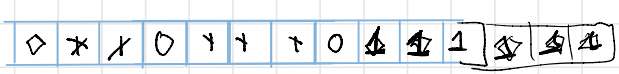


output string: n y 0 unary

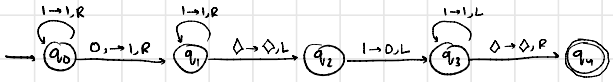
θ) $f(n,y) = ny$

n: 111 y: 11

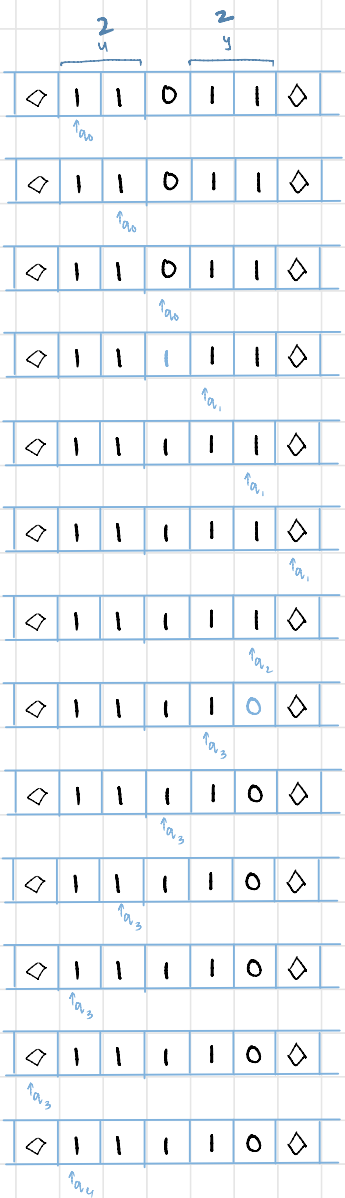




2. $f(x,y) = x+y$



$x = 11$, $y = 11$



Halt and Accept

combining Turing machine

↓ see slides

Expected Final Exam Questions

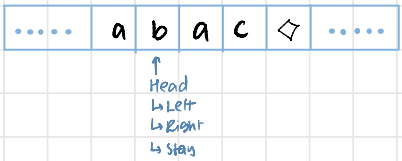
Design TM and explain the working of each state or combination of related states

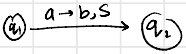
1. Binary Addition
2. Binary Subtraction $q_0\ waw \overset{*}{\boxtimes} q_f\ w$
3. Binary Multiplication

Give basic idea and Design TM over unary Alphabet and explain the working of each state or combination of related states

4. Primality Checking $q_0\ w \overset{*}{\boxtimes} q_f\ a$
5. Factorial Calculation $q_0\ w \overset{*}{\boxtimes} q_f\ w!$

Turing Machine with Stay Option

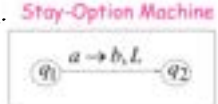




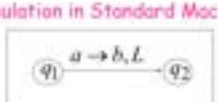
↳ Stay Option Machies are atleast as powerful as standard Machines

1. a stay option machine that never uses S move = standard machine
2. a standard machine can simulate a stay option machine

1. Stay-Option Machine



Simulation in Standard Machine



Similar for Right moves

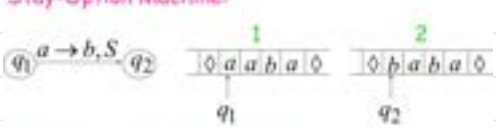
2. Stay-Option Machine

Simulation in Standard Machine

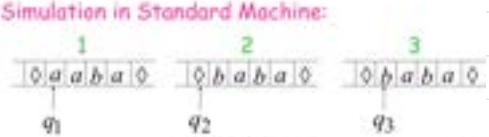
For every tape symbol x

Example

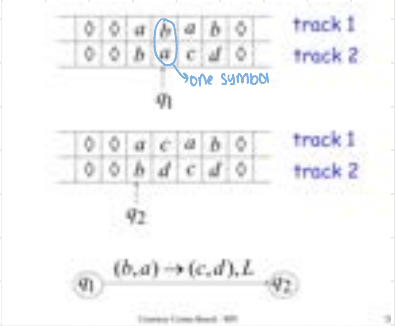
Stay-Option Machine:



Simulation in Standard Machine:



Turing Machine with Multiple Track Tape

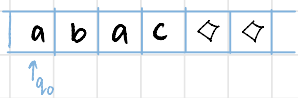


Turing Machine with Semi-Infinite Tape

↳ head extend infinitely only to the right

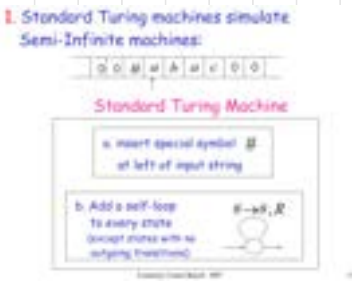
↳ Initial position is left most cell

↳ when head moves left from border, it returns to same position

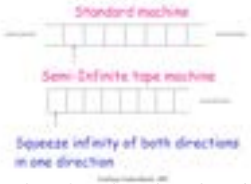


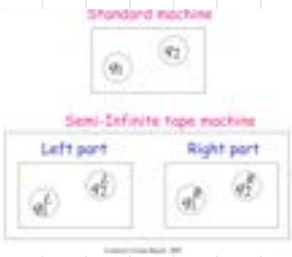
↳ Semi Infinite Machies have the same power as standard Machines

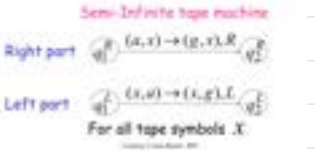
1. a standard machine can stimulate a Semi Infinite machine
2. a Semi Infinite machine can stimulate a standard machine

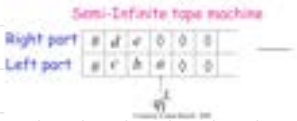


2. Semi-Infinite tape machines simulate Standard Turing machines

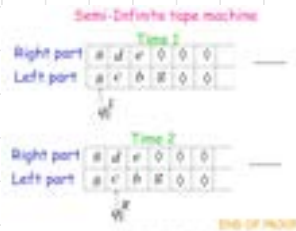






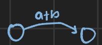


At the border:



Turing Machine with Off-Line

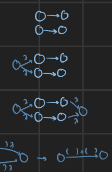
GENRALIZED TRANSITION GRAPH (GTG)

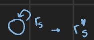
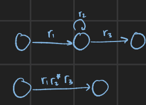
- ↳ initial state >0
- ↳ final state ≥0
- ↳ finite states, input letters 
- ↳ Directed edges connecting regular expression states

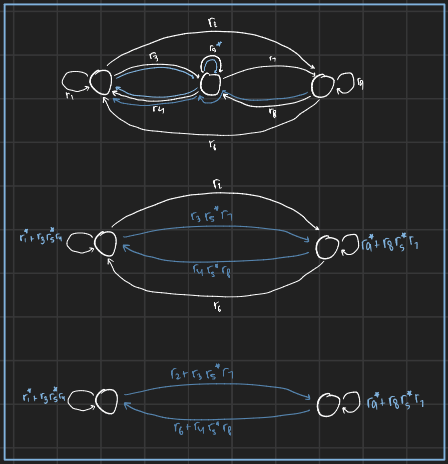
NON DETERMINISIM

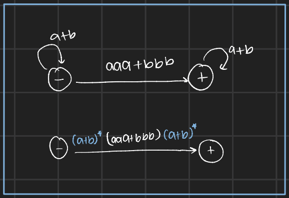
- ↳ TG, GTG
- ↳ There may exist none or more than one path for a certain string

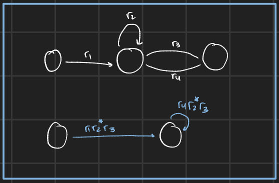
TG TO GTG CONVERSION



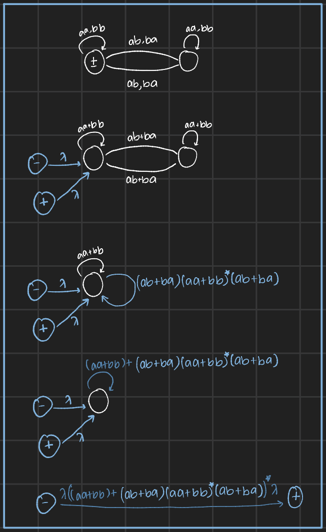
1. more than 1 start states = add new start state and connect old states with λ
2. more than 1 final states = add new final state and connect old states with λ
3. if a state has two or more edges = convert to 1 transition
   - ↳ loops = * 
   - ↳ 2 possibilities = + 
4. Bypass and state elimination 
   - ↳ if 3 states connected in sequence : eliminiate mid state and join the other two







since start, final state same
no other final state
add seperate start, final state and connect old state

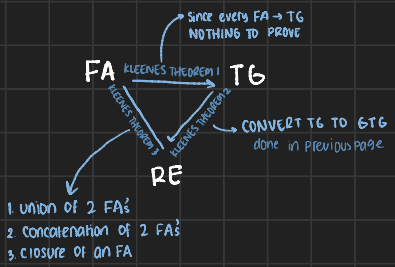


KLEENES THEOREMS

Regular Expressions

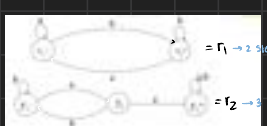
↳ FA
↳ TG
↳ RE

if can be expressed by one then the other and can be expressed aswell



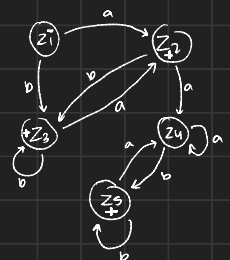
1. union of 2 FA's

↳ sum of 2 FA's = $r_1 + r_2$
↳ make transition table
↳ r_3 initial state = r_1, r_2 initial state
↳ r_3 final state = r_1, r_2 final state
↳ take $z_1 \equiv \{r_1 \text{ initial}^-, r_2 \text{ initial}^-\}$
↳ r_3 final = r_1 and r_2 finals
↳ total no of possible states = states of r_1 x states of r_2



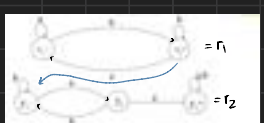
$r_3 = r_1 + r_2 \rightarrow$ 6 possible states

Old states	a	b
$z_1^- \equiv \{x_1^-, y_1^-\}$	$z_2 \equiv \{x_1^-, y_2\}$	$z_3 \equiv \{x_2^+, y_1^-\}$
$z_2 \equiv \{x_1^-, y_3^+\}$	$z_4 \equiv \{x_1^-, y_3^+\}$	$z_3 \equiv \{x_2^+, y_1^-\}$
$z_3^+ \equiv \{x_1^+, y_1^-\}$	$z_2 \equiv \{x_1^-, y_2\}$	$z_3 \equiv \{x_2^+, y_1^-\}$
$z_4^+ \equiv \{x_1^-, y_3^+\}$	$z_4 \equiv \{x_1^-, y_3^+\}$	$z_5 \equiv \{x_2^+, y_3^+\}$
$z_5^+ \equiv \{x_2^+, y_3^+\}$	$z_4 \equiv \{x_1^-, y_3^+\}$	$z_5 \equiv \{x_2^+, y_3^+\}$



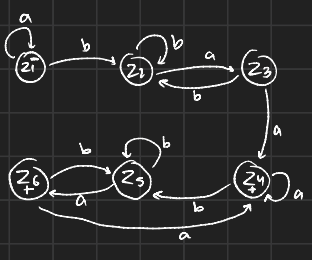
2 concatenation of 2 FA's

↳ $r_1 r_2$
↳ make transition table
↳ r_3 initial state = r_1 initial state
↳ r_3 final state = r_2 final state
↳ r_1 final state = r_2 initial state
↳ take $z_1 \equiv \{r_1 \text{ initial}^-\}$
↳ r_3 final = r_2 finals



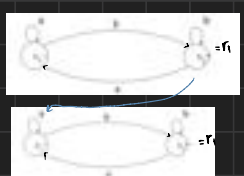
$r_3 = r_1 r_2$

Old states	a	b
$z_1^- \equiv \{x_1^-\}$	$z_1^- \equiv \{x_1^-\}$	$z_2 \equiv \{x_2^+, y_1^-\}$ (r_1 final, r_2 initial)
$z_2 \equiv \{x_2^+, y_1^-\}$	$z_3 \equiv \{x_2^+, y_2\}$	$z_2 \equiv \{x_2^+, y_1^-\}$
$z_3 \equiv \{x_2^+, y_2\}$	$z_4 \equiv \{x_2^+, y_3^+\}$	$z_2 \equiv \{x_2^+, y_1^-\}$
$z_4^+ \equiv \{x_2^+, y_3^+\}$	$z_4 \equiv \{x_2^+, y_3^+\}$	$z_5 \equiv \{x_2^+, y_2^-, y_3^+\}$
$z_5 \equiv \{x_2^+, y_2^-, y_3^+\}$	$z_6 \equiv \{x_2^+, y_2, y_3^+\}$	$z_5 \equiv \{x_2^+, y_2^-, y_3^+\}$
$z_6^+ \equiv \{x_2^+, y_2, y_3^+\}$	$z_4 \equiv \{x_2^+, y_3^+\}$	$z_5 \equiv \{x_2^+, y_2^-, y_3^+\}$



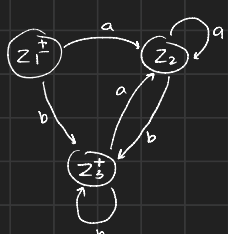
3. closure of an FA

↳ r^*
↳ make transition table
↳ r_1 final state = r_1 initial state
↳ r_3 initial = r_3 final ±
↳ take $z_1 = \{r_1 \text{ initial}^{\pm}\}$
↳ r_3 final = r_1 finals



$r_3 = r_1^*$

Old states	a	b
final → $z_1^{\pm} \equiv \{x_1^{\pm}\}$	$z_2 \equiv \{x_1^-\}$	$z_3 \equiv \{x_2^+, x_1^-\}$ (r_1 final, r_1 initial)
non final → $z_2 \equiv \{x_1^-\}$	$z_3 \equiv \{x_1^-\}$	$z_3 \equiv \{x_2^+, x_1^-\}$
$z_3^{\pm} \equiv \{x_2^+, x_1^-\}$	$z_2 \equiv \{x_1^-\}$	$z_3 \equiv \{x_2^+, x_1^-\}$



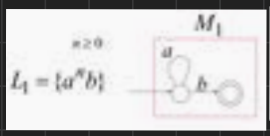
LECTURE 8

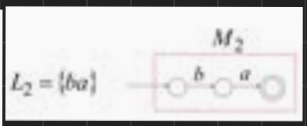
Properties of Regular Languages

- ↳ Union $L_1 \cup L_2$
- ↳ Concatenation $L_1 L_2$
- ↳ Star L_1^*
- ↳ Reversal L_1^R
- ↳ Complement $\overline{L_1}$
- ↳ Intersection $L_1 \cap L_2$

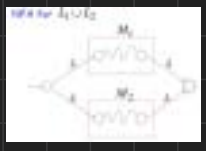
Any NFA converted to equivalent NFA with single final state

Show regular languages closed under

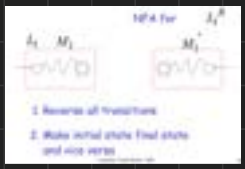


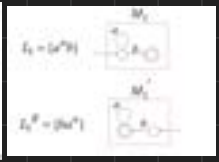


Union

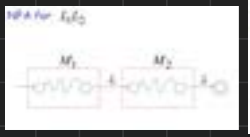


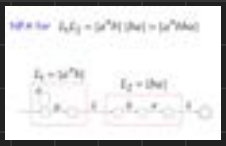
Reversal



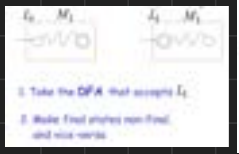


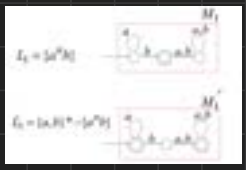
Concatenation



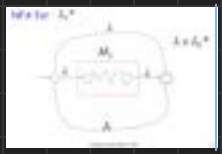


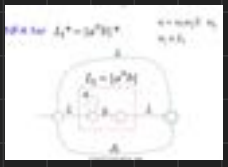
Complement



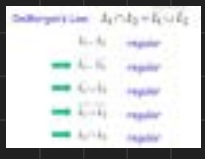


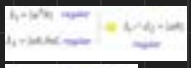
Star





Intersection





LECTURE 9

MINIMISING DFA's by PARTITION

Why Minimise DFA's

↳ Efficiency

↳ Better understanding of the language

↳ Compute similarity b/w the two FA's

↳ Determine if two DFA's recognise the same language

Minimization Process

↳ Remove inaccesible states

↳ Group equivalent states

↳ 2 states equal if all behaviors same

Minimization Steps

1. Places states of DFA into 2 equivalent classes

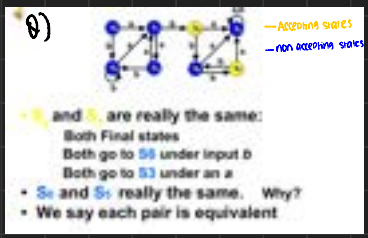
   ↳ final states
   ↳ non final states
   } 2 partitions

2. Repeat until no more change

   ↳ for each state in same partition

   if transition diff than those in it's group

   Partition to new equivalence class



1 final, non final partitions

	a	b
S_0	S_1	S_4
S_1	S_2	S_5
S_2	S_3	S_3
S_4	S_1	S_4
S_5	S_1	S_4
S_6	S_3	S_7
*S_2	S_3	S_6
*S_7	S_3	S_6

S_1, S_6 born end on final states

	a	b
S_0	S_1 I	S_4 I
S_1	S_2 I	S_5 II
S_2	S_3 I	S_3 I
S_4	S_1 I	S_4 I
S_5	S_1 I	S_4 I
S_6	S_3 I	S_7 II
*S_2	S_3 I	S_6 I
*S_7	S_3 I	S_6 I

S_0, S_4, S_5 same partition unlike S_7

	a	b
S_0	S_1 III	S_4 I
S_3	S_3 I	S_3 I
S_4	S_1 III	S_4 I
S_5	S_1 III	S_4 I
S_1	S_2 I	S_5 II
S_6	S_3 I	S_7 II
*S_2	S_3 I	S_6 III
*S_7	S_3 I	S_6 III

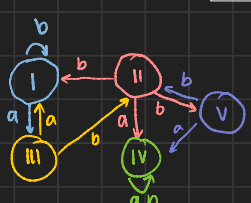
S_1, S_6 not same partion for a

	a	b
S_0	S_1 III	S_4 I
S_4	S_1 III	S_4 I
S_5	S_1 III	S_4 I
S_3	S_3 IV	S_3 IV
S_1	S_3 I	S_5 II
S_6	S_3 IV	S_7 II
*S_2	S_3 IV	S_6 III
*S_7	S_3 IV	S_6 III

	a	b
S_0	S_1 III	S_4 I
S_4	S_1 III	S_4 I
S_5	S_1 III	S_4 I
S_3	S_3 IV	S_3 IV
S_1	S_3 I	S_5 II
S_6	S_3 IV	S_7 II
*S_2	S_3 IV	S_6 V
*S_7	S_3 IV	S_6 V

	a	b
I	III	I
II	IV	V
III	I	II
IV	IV	IV
V	IV	II

5 partitions in order

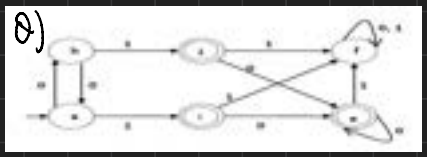


MINIMISING DFA's by Myphill-Nerode Theorem
TF ALGORITHM

Minimization Steps

1. Draw table for all pairs of state P,Q
2. Mark every pair in DFA where P→final, Q→not final, vice versa
   ↳ dosent have to be connected
3. Repeat until unable to mark more
   ↳ check for unmarked pairs $\delta(Q,x)$, $\delta(P,x)$
   UM ↳ unmarked pair → leave it as is
   DE ↳ pair dosent exist → leave it as is
   M ↳ marked pair → mark Q,P
4. Combine unmarked pairs and make single state

8)



Step 1, 2

	a	b	c	d	e	f
b						
c	✓	✓				
d	✓	✓				
e	✓	✓				
f	✓	✓	✓	✓	✓	

Step 3

unmarked pairs

(a,b) → $\delta(a,0)=b$ UM, $\delta(a,1)=c$ UM
$\delta(b,0)=a$ $\delta(b,1)=d$

(c,d) → $\delta(c,0)=e$ DE, $\delta(c,1)=f$ DE
$\delta(d,0)=e$ $\delta(d,1)=f$

(e,c) → $\delta(e,0)=e$ DE, $\delta(e,1)=f$ DE
$\delta(c,0)=e$ $\delta(c,1)=f$

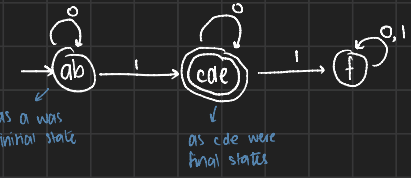
(e,d) → $\delta(e,0)=e$ DE, $\delta(e,1)=f$ DE
$\delta(d,0)=e$ $\delta(d,1)=f$

(f,a) → $\delta(f,0)=f$ UM, $\delta(f,1)=f$ M
$\delta(a,0)=b$ $\delta(a,1)=c$ M
so mark fa

(f,b) → $\delta(f,0)=f$ M
$\delta(b,0)=a$

Step 4

(a,b) (c,d) (e,c) (e,d)



as a was initial state

as cde were final states

LECTURE 10

PUMPING LEMMA

Used to prove that a language is not regular by contradiction

Pumping Lemma steps

1. Assume A is a regular language → for contradiction
2. Pumping length = ... → assume
3. Find string 's' in A such that $|s| \geq p$ → assume
4. Divide S into x y z
5. Consider cases where S can be divided to xyz
6. Show 3 Pumping conditions unsatisfied
   - ↳ 1. $xy^iz \in A$ for every $i \geq 0$
   - ↳ 2. $|y| > 0$
   - ↳ 3. $|xy| \leq p$
7. S cannot be pumped = CONTRADICTION

CAN NOT BE USED TO PROVE A LANGUAGE IS REGULAR

0) Prove $A = \{a^n b^n \mid n \geq 0\}$ is not regular

↳ Assume A is regular → for contradiction

Pumping length = p

$S = a^p b^p$ taking p = 7

x y z

aaaaaaa bbbbbbb

case 1: y is in the 'a' part

aaaaaaa bbbbbbb

x y z

$xy^2z \Rightarrow xy^2z$

aa aaaaaaaa abbbbbbb (repeat 2 for y^2)

a = b → as a^nb^n

1. $11 \neq 7$ ✗
2. $|y| = 4$ ✓
3. $|xy| = 6$ p = 7 ✓

case 2: y is in the 'b' part

aaaaaaa bbbbbbb

x y z

$xy^2z \Rightarrow xy^2z$

aaaaaaa bb bbbbbbbb b (y^2)

a = b

3. $7 \neq 11$ ✗
2. $|y| = 4$ ✓
3. $|xy| = 13$ p = 7 ✗

case 3: y is in the 'a' and 'b' part

aaaaa aabbbbbbb

x y z

$xy^2z \Rightarrow xy^2z$

aaaaa aabb aabb bbbbb (y^2)

pattern should be a's followed by b's

1. pattern not followed ✗
2. $|y| = 4$ ✓
3. $|xy| = 9$ p = 7 ✗

HENCE NOT REGULAR

Q) Prove $A = \{ yy \mid y \in \{0,1\}^* \}$ is not regular

Assume A is regular

Pumping length = P

$S = 0^P 1 0^P 1$ → choose any string

taking $P = 7$ → choose any y

case 1. y is in the '0' part

y^2

00 0000 0000 0100000001

1. doesn't follow pattern yy ✗
   - start and end pattern should be the same
2. $|y| = 4$ ✓
3. $|xy| > 6$ $P = 7$ ✓

Hence not regular

DFA WITH OUTPUTS

Output
- depends on input
- doesn't depends on input

Mealy State Machines

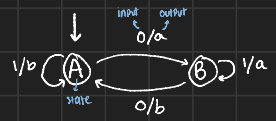
↳ Output depends on current state and input

↳ inputs = outputs

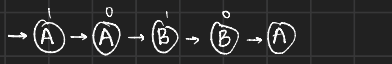
↳ no final state

$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

- Q: finite set of states
- Σ: inputs
- δ: transition function $Q \times \Sigma \to Q$
- λ: output function $\Sigma \times Q \to \Delta$
- q_0: initial state



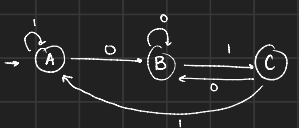
e.g. 1010

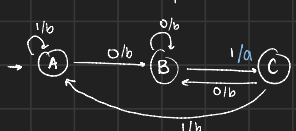


Q) construct mealy machine that produces a when 01 is input string

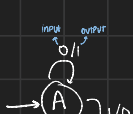
$\Sigma = \{0,1\}$

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





Q) construct mealy machine that produces 1's complement



1 0 1 0 0

0 1 0 1 1

Moore State Machines

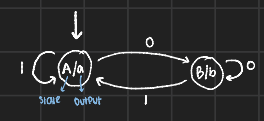
↳ Output depends on current state

↳ inputs = outputs + 1

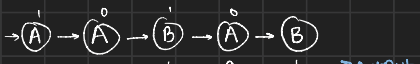
↳ no final state

$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

- Q: finite set of states
- Σ: inputs
- δ: transition function $Q \times \Sigma \to Q$
- λ: output function $Q \to \Delta$
- q_0: initial state



e.g. 1010

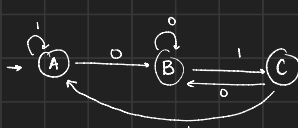


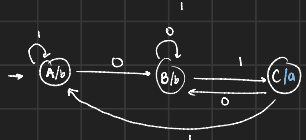
directly comes without any input

Q) construct moore machine that produces a when 01 is input string

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

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





context free languages

context free grammars

$G=(V,T,S,P)$

- variables
- terminal symbols
- start variable
- production rule: $A \to a$, $a \in \{V \cup \Sigma\}^*$, $A \in V$

Q) generate equal no of a's and b's in the form $a^n b^n$ (not regular lang)

$G = \{ (S,A), (a,b), (\overbrace{S \to aAb,\ A \to aAb \mid \epsilon}^{P}) \}$ (epsilon = empty symbol)

$S \to a\underline{A}b$ (replace $A \to aAb$) $V: \{S, A\}$

$\to aa\underline{A}bb$ (replace $A \to aAb$) $T: \{a, b\}$

$\to aaa\underline{A}bbb$ (replace $A \to \epsilon$)

$\to aaabbb$

$\to a^3b^3 \to a^nb^n$

NOTATION

- beg lower case letters: a,b,c ... Terminal symbols
- end lower case letters: w,x,y... strings of terminals
- beg upper case letters: A,B,C ... variables
- end upper case letters: X, Y, ... Terminals/strings
- lower case Greek letters: α, β, γ... Strings of Terminals/variables

Derivation order

1. $S \to AB$ 2. $A \to aaA$ 3. $A \to \lambda$
4. $B \to Bb$ 5. $B \to \lambda$

↳ Leftmost derivation order of string

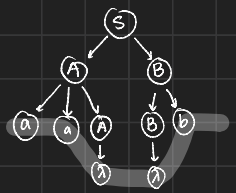
$S \xrightarrow{1} AB \xrightarrow{2} aaAB \xrightarrow{3} aaB \xrightarrow{4} aaBb \xrightarrow{5} aab$

↳ Rightmost derivation order of string

$S \xrightarrow{1} AB \xrightarrow{4} ABb \xrightarrow{3} Ab \xrightarrow{2} aaAb \xrightarrow{3} aab$

Derivation Trees

$S \to AB$ $A \to aaA \mid \lambda$ $B \to Bb \mid \lambda$



= aab

↳ when 2 or more derivation trees then context free grammar is ambiguous

is this comming?

push down automata

not in Mid 2

↳ not on this page