Theory of Automata

Languages

Formal — Informal

Syntactic — Semantic

↳ structure

↳ logic focused

ALGOL : has 113 letters

automata: something that works automatically

Strings

concatenation of finite symbols from alphabet

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

Empty/NULL string (λ / Λ)

Alphabets (Σ)

↳ finite non empty set of symbols

↳ letters, digits, GOTO, IF, operators

↳ $\Sigma = \{a, b\}$

Words

↳ strings belonging to language

↳ if ✓

↳ ifff ✗

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

↓ concatenation

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 ($\text{Rev}(s) / s^r$)

↳ reverse words order → not letter

$s = BaBbabBd$

$\text{Rev}(s) = d\ B\ bab\ aB\ B$

Concatenations

↳ $w^n = \overbrace{ww \ldots w}^{n}$

$abba^2 = abbaabba$

↳ $w^0 = \lambda$ → NULL/EMPTY string

$abba^0 = \lambda$

* Operation

↳ all possible strings from alphabet

↳ all possible combinations

↳ includes null too

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

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

+ Operation

↳ all possible combinations

↳ dosent includes null too

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

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

language

alphabet ↑

any subset of Σ^*

$\{\lambda\}$ → diff language

$\{a, aa, aab\}$ → diff language

words

order dosent 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

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

- a^n: exactly n
- infinite language
- $n \geq 0$: can be zero so null string include
- 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
- $L^R = \{w^R : w \in L\}$, $\Sigma = \{a, b\}$

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

  $\{a, \underline{aBb}\}^R = \{a, \underline{ba}B\}$

  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

- $(a+b.c)^* = \{\lambda, a, bc, abc, bca, aa, \ldots\}$
- $a^+ = aa^*$ (will always come so no λ) $= \{a, aa, \ldots\}$
- $(ab)^* = \{\lambda, ab, abab, \ldots\}$ → 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
- $(b^* a b^*) = \{a, bab, ba, ab, bbab\}$ (no order) → only 1 a exists
- $(ab^*) = \{a, ab, abb\}$ → only 1 a but at the start
- $(\underline{b^* a} + \underline{ba^*})^* = \{a, b, bba, baa\}$ (choose any time, since no star outside)
- $(ab)^2 = \{\underline{abab}\}$ (length)
- $((a+b)(a+b))^* = \{\lambda, aa, ab, ba, bb\}$ → 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

$A = \{ Q, \Sigma, \delta, q_0, F \}$ → 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

cuz finite state

$\Sigma = \{ \lambda, l, m, lm \}$ $(\lambda + l + m + lm)$

↳ automata starting with a

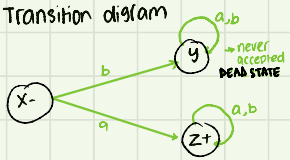
↳ Transitions table

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

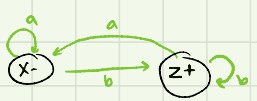
$a(a+b)^*$

↳ 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

$(a+b)^* b$

$Q = \{x, y, z\}$

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

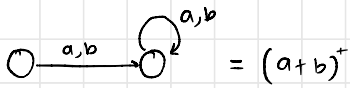
$q_0 = X$

$F = Z$

δ :

Transition Graph

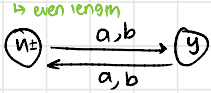
@ rigix



$= (a+b)^+$

	a	b
X±	y	y
y	x	x

↳ ± accepts λ
↳ even length

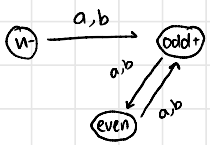


$((a+b)(a+b))^*$

even

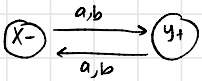
	a	b
X-	odd	odd
odd+	even	even
even	odd	odd

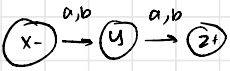
↳ no ± as it accepts λ = even



odd

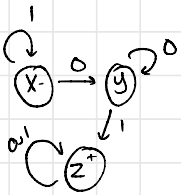
$((a+b)(a+b))^*(a+b)$



1. 

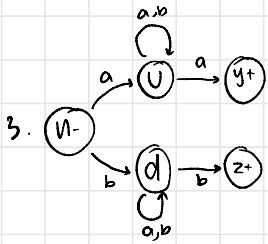
$(a+b)(a+b)(a+b)$

odd and 3 length

2. 

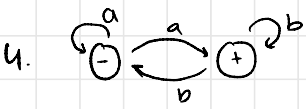
$(0+1)^*01(0+1)^*$

x 01 y

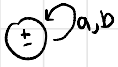
3. 

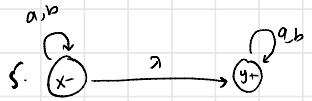
$a(a+b)^*a + b(a+b)^*b$

start end same letter

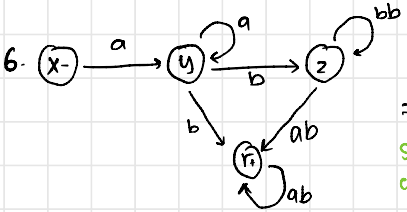
4. 

$(a+b)^*$



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 

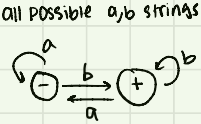
TG

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

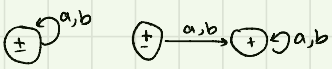
NFA NON DETERMINISTIC FINITE AUTOMATA

- 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 

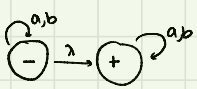
all possible a,b strings

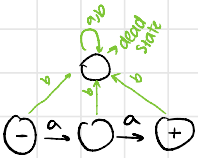


all possible a,b strings



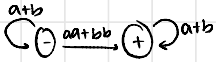
all possible a,b strings





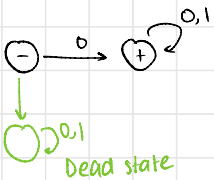
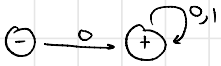
GTG

- TG
- regular expressions



NFA-Λ

if asked to make FA → NFA / DFA → usually

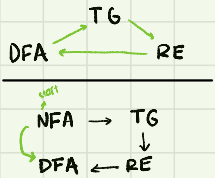
DFA	all string starting with 0	NFA
		

initial state → ○ — final state ◎

⊖ — ⊕

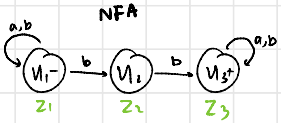
DFA → NFA DFA → NFA-Λ

NFA → TG NFA-Λ → TG



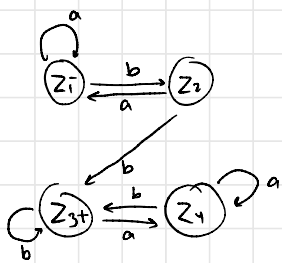
NFA TO DFA

NFA

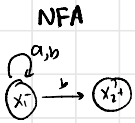


DFA

old states	a	b
$z_1^- \equiv \{x_1^-\}$	$z_1^- \equiv x_1$	$z_2 \equiv \{x_1, x_2\}$ (can put x_1^- as minor)
$z_2 \equiv \{x_1, x_2\}$	$z_1 \equiv \{x_1, \emptyset\}$	$z_3^+ \equiv \{x_1, x_2, x_3\}$
$z_3^+ \equiv \{x_1, x_2, x_3\}$	$z_4^+ \equiv \{x_1, x_3\}$ (contains x_3^+ final state)	$z_3^+ \equiv \{x_1, x_2, x_3\}$
$z_4^+ \equiv \{x_1, x_3\}$	$z_4^+ \equiv \{x_1, x_3\}$	$z_3^+ \equiv \{x_1, x_2, x_3\}$

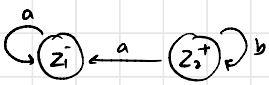


NFA



DFA

Old states	a	b
$z_1^- \equiv x_1^-$	$z_1^- \equiv x_1^-$	$z_2^+ \equiv \{x_1^-, x_2^+\}$
$z_2^+ \equiv \{x_1^-, x_2^+\}$	$z_1^- \equiv x_1^-$	$z_2^+ \equiv \{x_1^-, x_2^+\}$



EPSILON MISSING

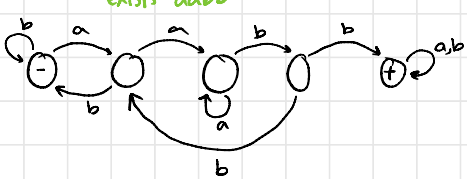
8) dosent have aabb DFA

↳ first make does have aabb

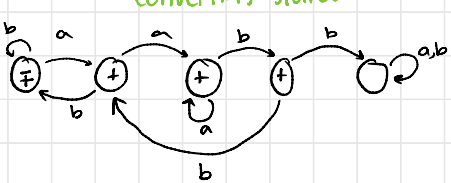
↳ convert final state to non final state

↳ 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

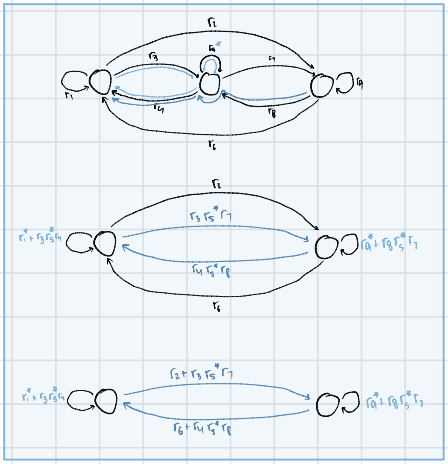
NON DETERMINISM

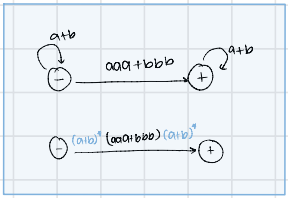
- ↳ TG, GTG
- ↳ There may exist none or more than one path for a certain string

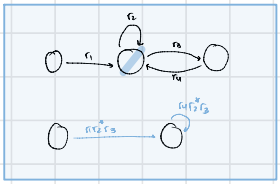
TG TO GTG CONVERSION

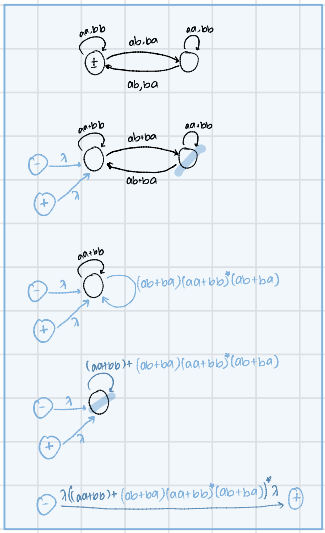
1. more than 1 start states = add new start state and connect old states with λ
2. more than 1 final states = add new final state and connect old states with λ
3. if a state has two or more edges = convert to 1 transition
   - ↳ loops = *
   - ↳ 2 possibilities = +
4. Bypass and state elimination
  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

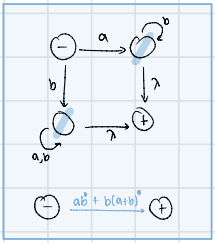
→ since start and final state same
→ and no other final state exists
add seperate start and final state and connect to old state

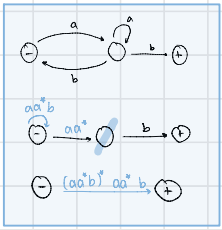










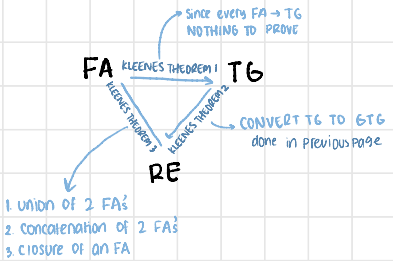


KLEENES THEOREMS

Regular Expressions

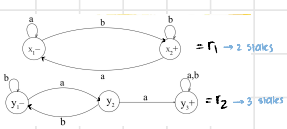
↳ FA
↳ TG
↳ RE

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



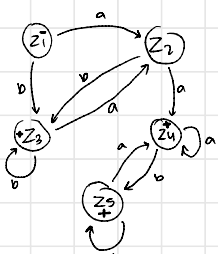
1. union of 2 FA's

↳ 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



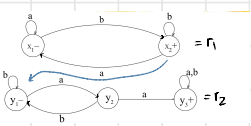
$r_3 = r_1 + r_2$ → 6 possible states

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



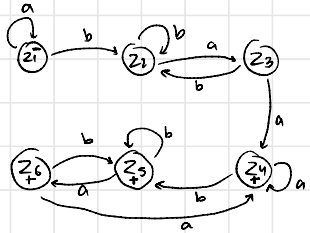
2. concatenation of 2 FA's

↳ $r_1 r_2$
↳ 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 $z_1 \equiv \{r_1 \text{ initial}^-\}$
↳ r_3 final = r_2 finals



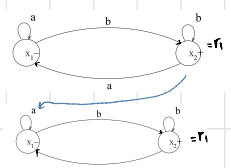
$r_3 = r_1 r_2$

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)
$z_2 \equiv \{x_2^+, y_1^-\}$	$z_3 \equiv \{x_1^-, y_2\}$	$z_2 \equiv \{x_2^+, y_1^-\}$
$z_3 \equiv \{x_1^-, y_2\}$	$z_4 \equiv \{x_1^-, y_3^+\}$	$z_2 \equiv \{x_2^+, y_1^-\}$
$z_4^+ \equiv \{x_1^-, y_3^+\}$	$z_4 \equiv \{x_1^-, y_3^+\}$	$z_5 \equiv \{x_2^+, y_3^+\}$
$z_5^+ \equiv \{x_2^+, y_3^+\}$	$z_6 \equiv \{x_1^-, y_2, y_3^+\}$	$z_5 \equiv \{x_2^+, y_3^+\}$
$z_6^+ \equiv \{x_1^-, y_2, y_3^+\}$	$z_4 \equiv \{x_1^-, y_3^+\}$	$z_5 \equiv \{x_2^+, y_3^+\}$



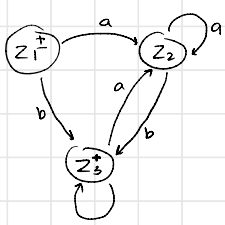
3. closure of an FA

↳ 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



$r_3 = r_1^*$

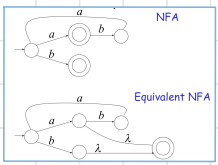
Old states	a	b
final → $z_1^\pm \equiv \{x_1^\pm\}$	$z_2 \equiv \{x_1^-\}$	$z_3 \equiv \{x_2^+, x_1^-\}$ (r_1 final, r_1 initial)
non final → $z_2 = \{x_1^-\}$	$z_1 = \{x_1^\pm\}$	$z_3^+ \equiv \{x_2^+, x_1^\pm\}$
$z_3^+ \equiv \{x_2^+, x_1^\pm\}$	$z_2 \{x_1^-\}$	$z_3^+ \equiv \{x_2^+, x_1^-\}$

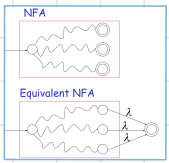


Properties of Regular Languages

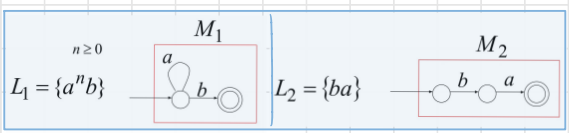
- Union $L_1 \cup L_2$
- Concatenation $L_1 L_2$
- Star L_1^*
- Reversal L_1^R
- Complement $\overline{L_1}$
- Intersection $L_1 \cap L_2$

Any NFA converted to equivalent NFA with single final state

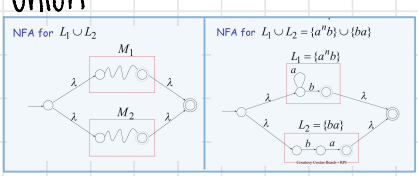




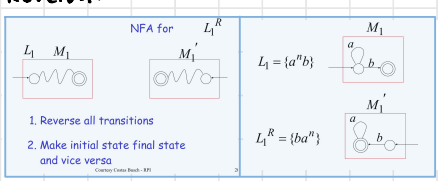
Show regular languages closed under



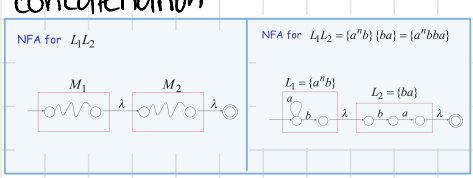
Union



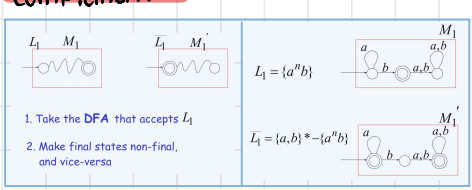
Reversal



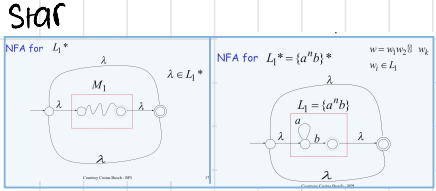
Concatenation



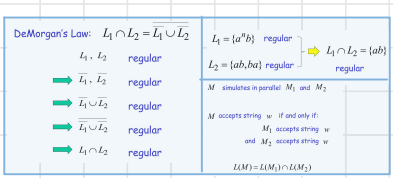
Complement



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

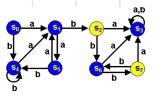
↳ 2 states equal if all behaviors same

Minimization Steps

1. Places states of DFA into 2 equivalent classes
   - ↳ final states
   - ↳ non final states

   } 2 partitions
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_2 and S_4 are really the same:
  - Both Final states
  - Both go to S6 under input *b*
  - Both go to S3 under an *a*
- S_0 and S_5 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_2
S_3	S_5	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_5	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_2 II
S_3	S_5 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_5 I	S_6 I

S_0, S_4, S_5 same partition unlike S_3

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

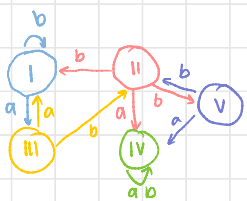
S_1, S_6 not same partion for a

	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_2 II
S_6	S_3 IV	S_7 II
*S_2	S_3 IV	S_6 III
*S_7	S_5 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_2 II
S_6	S_3 IV	S_7 II
*S_2	S_3 IV	S_6 V
*S_7	S_5 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

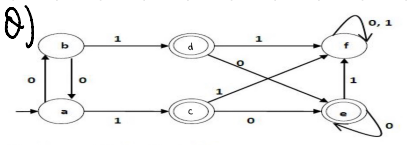


MINIMISING DFA's by Myphill-Nerode Theorem
TF ALGORITHM

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

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

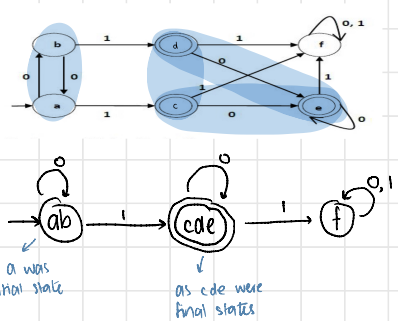
(e,d) → $\delta(e,0)=e$ DE, $\delta(e,1)=f$ DE
$\delta(d,0)=e$ $\delta(d,1)=f$

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



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

   $|s| \geq P$ → 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^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

aaaaaaabbbbbbb

x y z

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

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

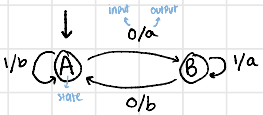
↳ inputs outputs
$n = n$

↳ no final state

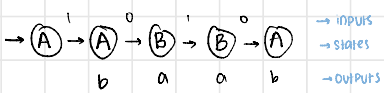
$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

- 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

NO FINAL STATE

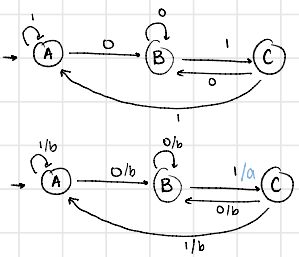


eg 1010

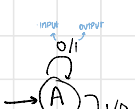


Q) construct mealy machine that produces a when 01 is input string

$\Sigma = \{0,1\}$
$\Delta = \{a,b\}$



Q) construct mealy machine that produces 1's complement



10100
01011

↳ aka complementing machine

Moore State Machines

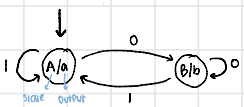
↳ Output depends on current state

↳ inputs outputs
$n = n+1$

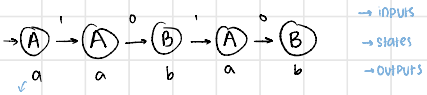
↳ no final state

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

- 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



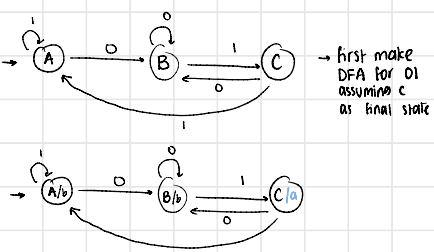
eg 1010



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

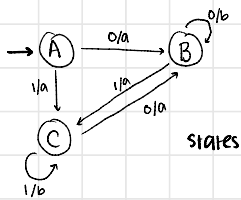
↳ no of states increased — max no of moores state = mealy states × outputs

↳ output of the state = incoming edges output

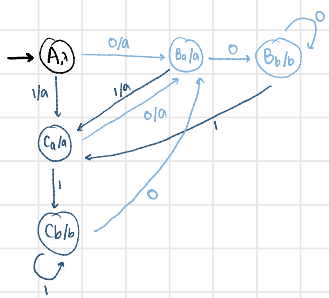
↳ if state already has a diff output

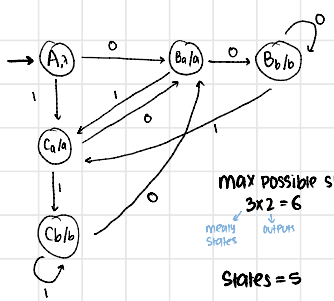
↳ create new state

↳ according to outputs make edges for new states



states = 3





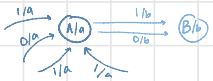
max possible states
3 x 2 = 6
mealy states × outputs

states = 5

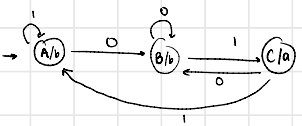
Moore → Mealy

↳ no of states same

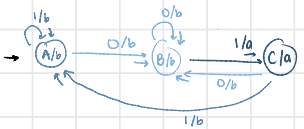
↳ incoming edges output = output of the state



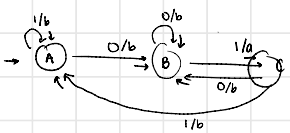
Q)



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

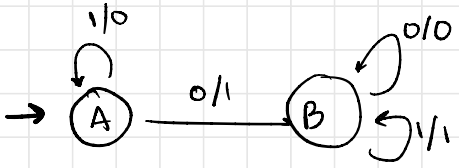


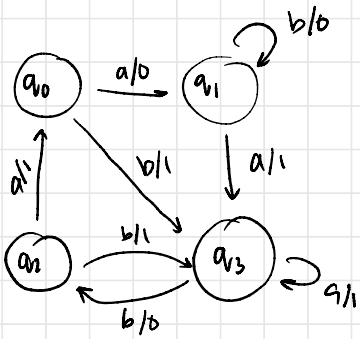
Old state	0	1
→ A	B/b	A/b
B	B/b	C/a
C	B/b	A/b

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

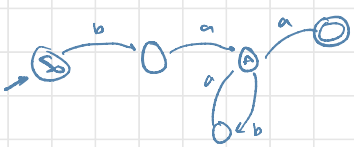
$S \to Aab$

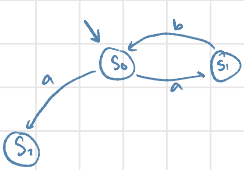
$A \to Aab \mid B$

$B \to a$

$S \to baA$

$A \to baA \mid a$





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

1) $S \to aSb|Ab|\lambda$, $A \to aAb|\lambda$ ✓
2) $S \to SS|aSb|bSa|\lambda$ ✗ $L(G) = \{w : n_a(w) = n_b(w)\}$ (number of a in string w)
3) $S \to A$, $A \to aB|\lambda$, $B \to Ab$ ✓ $L(G) = \{a^n b^n : n \geq 0\}$

Right Linear Grammar

↳ all productions form
- ↳ no variable
- ↳ variable but only in the end

$A \to xB$ or $A \to x$ (string of terminals)

1) $S \to abS$
2) $S \to a$

Left Linear Grammar

↳ all productions form
- ↳ no variable
- ↳ variable but only in the start

$A \to Bx$ or $A \to x$ (string of terminals)

1) $S \to Aab$
2) $S \to Aab|B$
3) $S \to a$

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 $L_1 \cap L_2$

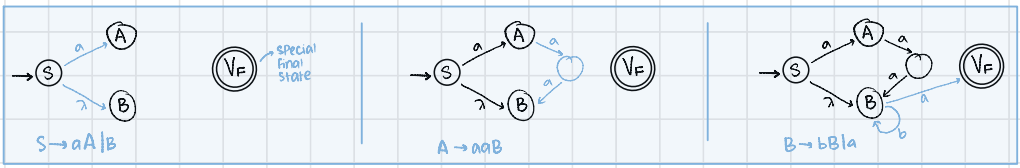
1) $G = \{S \to abS|a\}$
   $L(G) = (ab)^*a$

2) $G = \{S \to Aab , A \to Aab|B , B \to a\}$
   $L(G) = aab(ab)^*$

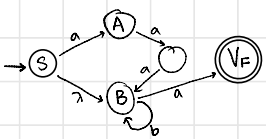
Proof Right Linear Grammar is Regular → part 1

↳ 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

$L(G') \rightarrow L(G')^R \rightarrow L(G)$
Regular Language → Regular Language → Regular Language

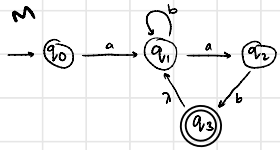
0) G = { A → abC_k | v } → left linear G
G' = { A → kCba | v^R } → right linear G

Proof Regular Language is Linear Grammar → part 2

↳ construct from M a Regular grammar G

↳ such that L(M)=L(G)

Q) M

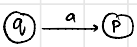


$L = ab^*ab(b^*ab)^*$

$L = L(M)$

convert to Right Linear Grammar

1. For any transition



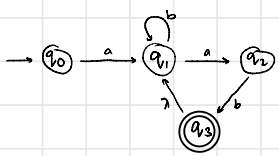
$q \longrightarrow aP$

variable — terminal — variable

2. For any final transition

$q_f \longrightarrow \lambda$

Q)



$q_0 \rightarrow aq_1$

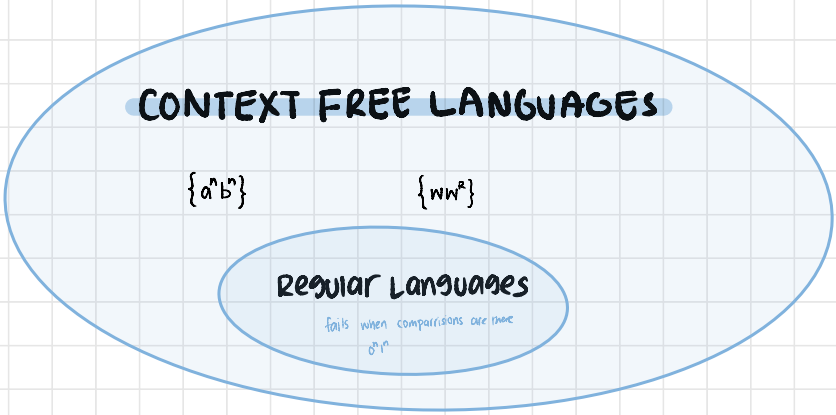
$q_1 \rightarrow bq_1$

$q_1 \rightarrow aq_2$

$q_2 \rightarrow bq_3$

$q_3 \rightarrow q_1$

$q_3 \rightarrow \lambda$



context free languages

context free Grammar

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

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

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

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

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

$\to aaabbb$

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

$V=\{S,A\}$

$T=\{a,b\}$

CFL to CFG

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

$S\to aSb \mid \lambda$

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

$S\to aSb \mid bb$

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

$S\to aaSb \mid \lambda$

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

$S\to aaSb \mid aaa$

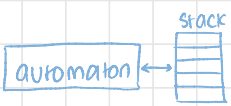
5) a^mb^n, $m\geq n$

$S\to AS_1$

$S_1\to aS_1b \mid \lambda$

$A\to aA \mid a$

Pushdown Automata



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

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

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

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

8) $a^mb^mc^n$ $m,n\geq 0$

$S\to S_1C$

$S_1\to aS_1b \mid \lambda$

$C\to cC \mid \lambda$

Context Free Grammar

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

- V: variables
- T: Terminal symbols
- S: start variable
- P: Production rule

$A \to a$

$a=\{V \cup \Sigma\}^*$

$A \in V$

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

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

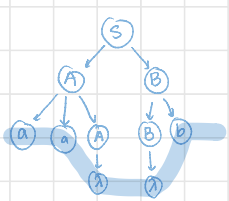
↳ Rightmost derivation order of string

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

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

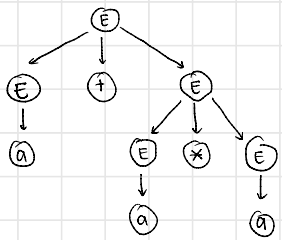
↳ two left most derivation trees

Q) E → E+E | E*E | E | a

a + a * a

2 + 2 × 2

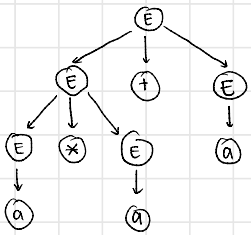
left most derivation



2 + 2 * 2 = 6 → CORRECT

E → E+E → a+E → a+E*E → a+a*E → a+a*a

left most derivation



2 + 2 * 2 = 8

E → E*E → E+E*E → a+E*E → a+a*E → a+a*a

2 left most derivation tress hence ambigious

How to remove Ambiguity

↳ it is difficult to achive and sometimes possible

Q) E → E+E | E*E | E | a → Ambigious Grammar

CONVERT

E → E+T | T , T → T*F | F , F → E | a → non Ambigious Grammar

Inherent Ambiguity

↳ will always have ambigious grammar

Example: $L = \{a^n b^n c^m\} \cup \{a^n b^m c^m\}$

$S \to S_1 \mid S_2 \quad S_1 \to S_1 c \mid A \quad S_2 \to aS_2 \mid l$

$A \to aAb \mid \lambda \quad B \to bBc \mid \lambda$

Courtesy Costas Busch - RPI

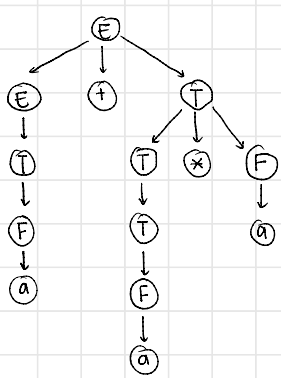
6

$L = \{a^n b^n c^m\} \cup \{a^n b^m c^m\}$

$n, m \geq 0$

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 → aSb
S → λ
S → A
A → aA
↳ useless production

Q) S → aSb
S → λ
S → A
A → aA → USELESS
B → C
C → D

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 → aS | A | C
A → a
B → aa
C → aCb

→ ONLY TERMINALS ROUND 1 {A, B, S}

S → aS | A
A → a
B → aa

→ variables rechable from S

S → aS | A
A → a

Simplication of CFG

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

Q) S → aB
A → aaA
A → abBc
B → aA
B → b

Substitute B → b

S → aB | ab
A → aaA
A → abBc | abbc
B → aA

Substitute B → aA

S → aB | ab | aaA
A → aaA
A → abBc | abbc | abaAc

S → ab | aaA
A → aaA
A → abbc | abaAc

→ Equivalent Grammar
use substitution rules