Discrete Maths

5

(categories)
Based on type of production rules that is how does terminals and non-terminals are arranged

~~there~~ are types of grammar ~~observed~~ by

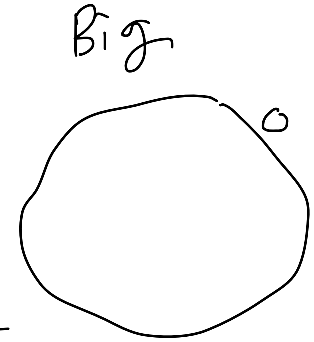
Chomsky.

He actually is looking for "Universal Grammar"

Hypothetically, humans and alien can also communicate.

Rules :

(no restrictions) type-0



Type-1

$\alpha \to \beta$

i.e $aA \to a$ ✗

$aAb \to aBCb$

length of α $\leq$ length of β

$A \to ab$

i.e ✓ $A \to aA$

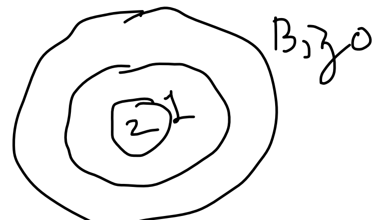
$aAb \to aBCb$, $3<4$

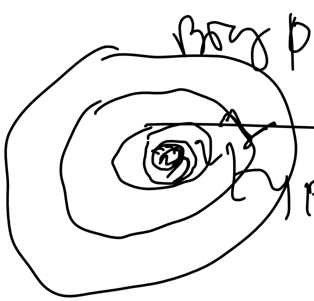
α, β arbitrary strings of terminals and non-terminals

Type-2

$A \to \alpha$

left always NT single





Type-3

$A \to a \mid aB$ } same

$A \to a \mid Ba$

$a \in T$

$B \in N$

A language is said to be a type-i

i (0,1,2,3) language if it can be

represented (specified) by a type-i grammar.

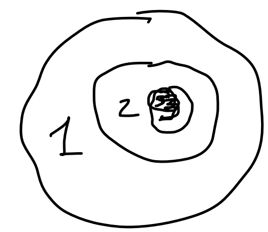
BUT

Can NOT be specified by

a type (i+1) grammar.

i.e. $L = \{ a^k b^k \mid k \geq 1 \}$

Big O



$A \rightarrow aAb$

$A \rightarrow ab$

$A \rightarrow \alpha$

Type-2 language. $S \rightarrow aA$, $S \rightarrow Aa$

but not type-3 (regular)

bcoz right hand more than 2 length.

Phrase structure grammar — PSG

is based on constituency relation.

is type-0 grammar.

"Flexible" / open

Type-0 language can be utilized / represents

Turing machine

Turing machine

≃ A computer that is capable of ~~running~~ program

storage + processing

constituency

vs

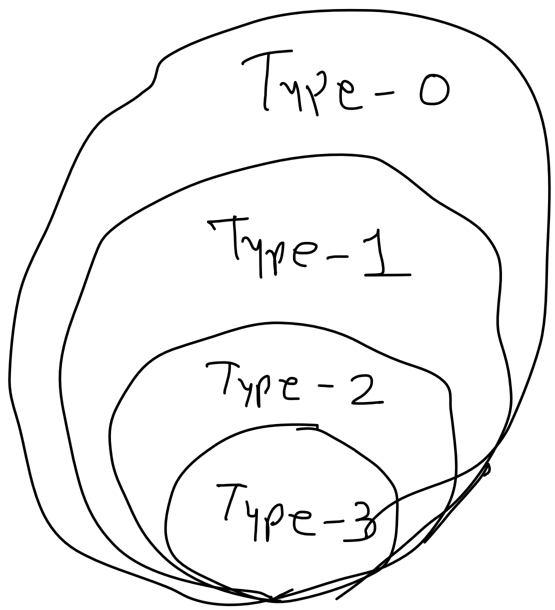
Dependency relation

Type-3 / Regular language

is for FSA

Grammar:

specific rules that determine input strings are accepted (part of language) or rejected (not part of language) class / types of grammar and related machines



Turing machine

Linear bounded automation

pushdown Automata
(stack memory)

FSA/FSM
(finite memory
direct/indirect)

$a^n b a^n$?

$a^n b^n c^n$

&

$a^* b c^*$

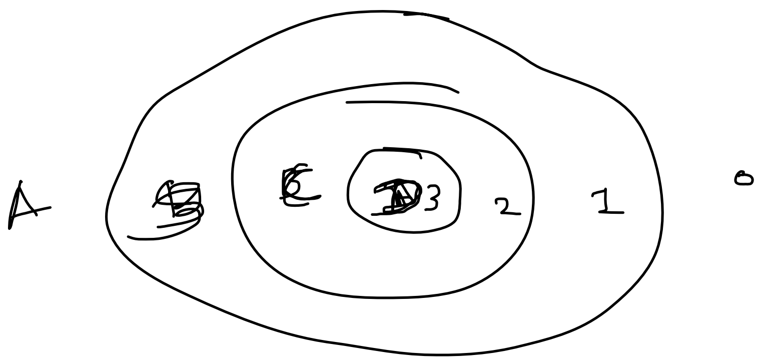
aaaa bccc ..

a^* ↑ c^*

$a^n b c^n$ or $a^n c^n$??

All type-0 grammar
are also
type 1, 2, 3. ?

All type-3 grammar are also
type 2, 1, 0 ?



A contains B, C, D

B contains C, D

C contains D

All type-0 are 1, 2, 3

~~All~~ type-1 are 2, 3

All type-2 are 3

OR

All type-3 are 2, 1, 0

All type-2 are 1, 0

All type-1 are 0

type-0 grammar are

used by Turing machine

"General purpose"

type-1

context sensitive grammar

If there is a production of the form

$A r \to l \alpha r$ (different than $A \to \alpha$)

$\alpha \to \beta$

Observe context sensitiveness

A can be replaced by α only

when it is having context exact

that is it is surrounded by

the strings l and r, where l is left

context, r is right side context.

Context free grammar (type-2)
are used in defining the
syntax of almost all programming
languages. (context free grammar)

Regular grammars (type-3) can be
used to search text /pattern matching.

P.S. regular expressions

Example 30 cents vending machine

Total Deposits	New 5¢	New 10¢	New 25¢	merchandise delivered	Total deposit
0¢	0+5 => 5	0+10 => 10	0+25 => 25	NOTHING	0¢
5¢			30 or more	NOTHING	5¢
10¢			30 or more	NOTHING	10¢
15¢			30 or more	NOTHING	15¢
20¢		30 or more	30 or more	NOTHING	20¢
25¢	30¢ or more	30¢ or more	30¢ or more	NOTHING	25¢
30¢ or more	5¢	10¢	25¢	Bubble Gum	30¢ or more
state	New total deposit (a) Input			(b) output	State

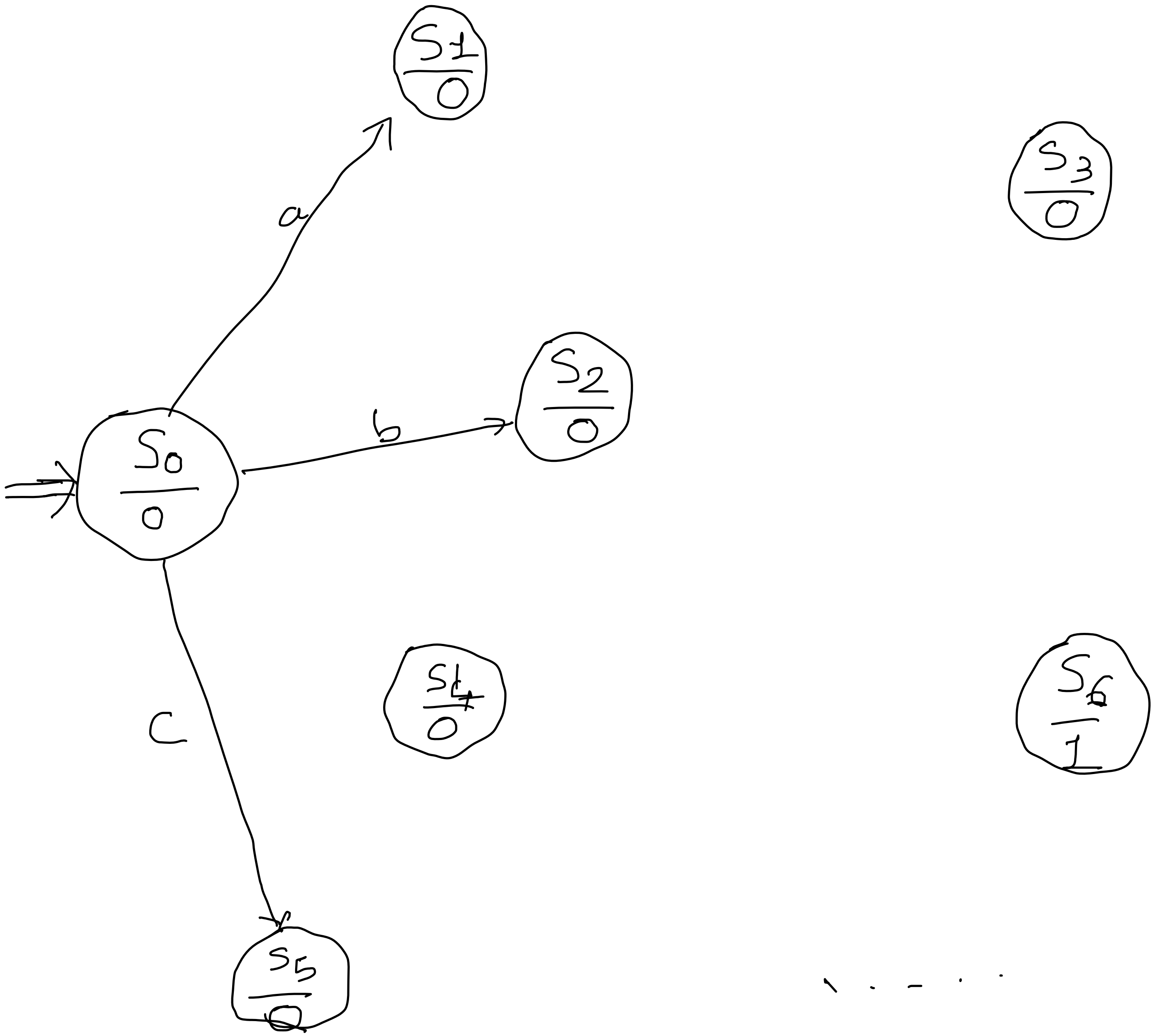
output is attached with state ~~and~~ not transition.

Input

indicate start

State	a	b	c	output
⇒ S_0	S_1	S_2	S_5	0
S_1	S_2	S_3	S_6	0
S_2	S_3	S_4	S_6	0
S_3	S_4	S_5	S_6	0
S_4	S_5	S_6	S_6	0
S_5	S_6	S_6	S_6	0
S_6	S_1	S_2	S_5	1

If You reach to ~~state~~ S_6, You receive bubblegum.



S1
0
a
S3
0
S2
0
b
S0
0
c
S4
0
S6
1
S5
0

FSM as models of physical systems

Example:

Design a modulo 3 counter

Input: sequence of 0's & 1's & 2's

Output: sequence of 0s, 1s and 2s

such that at any instant

The output is

equal to the modulo 3

sum of the digits in

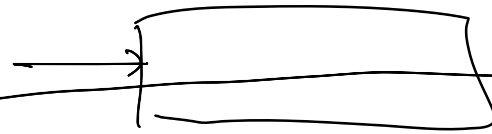
the input sequence.

State A :

situation that
modulo 3 sum of all input
digits is 0

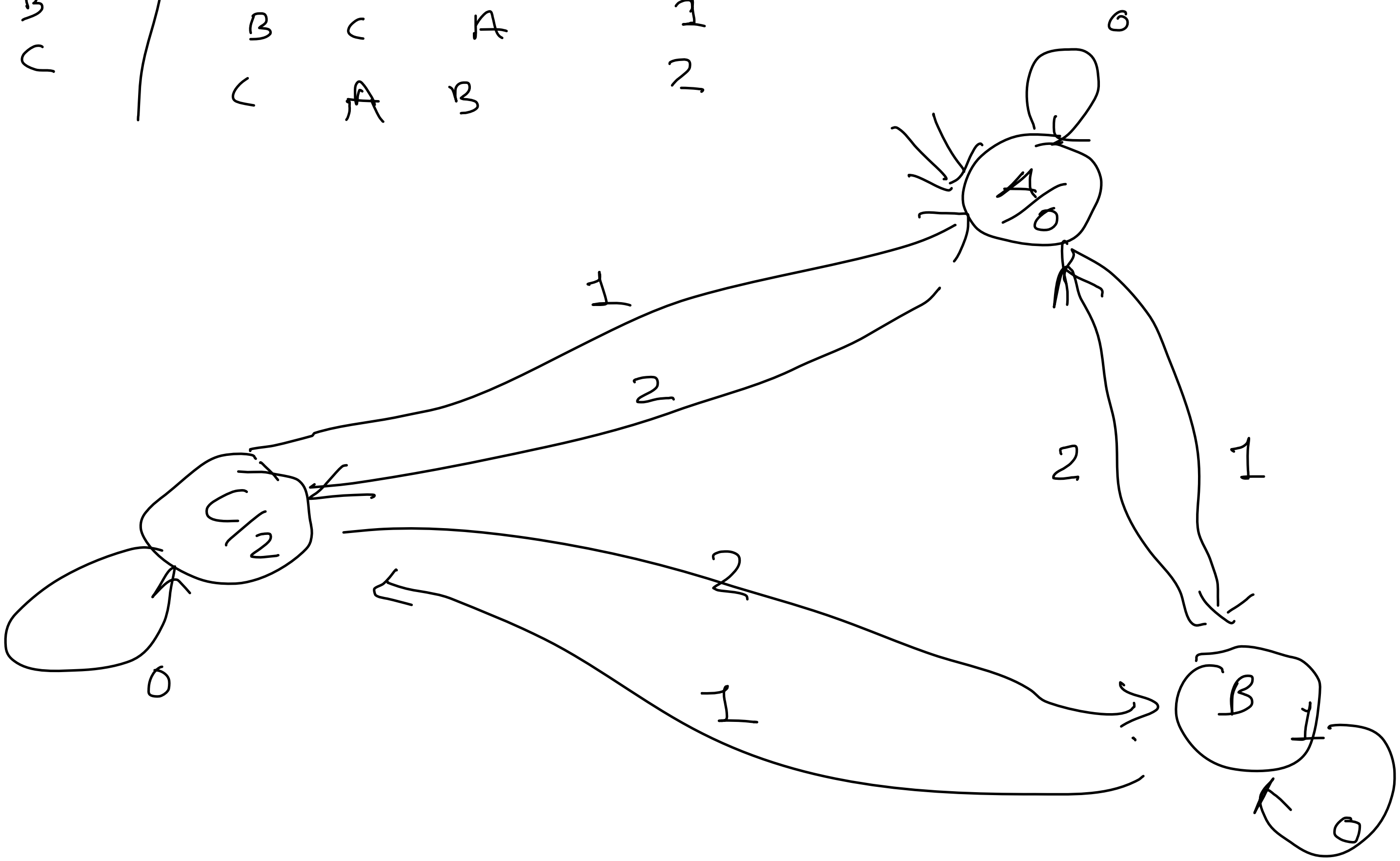
State B :
modu~~lo 3 sum of all input~~
digits is 1

State C :
modulo 3 sum of all digits
is 2



	sum	output	
0	0	0	
0	0	0	
1	1	1	
1	2	2	
1	3	0	∵ 3 % 3 => 0
2	5	2	∵ 5 % 3 => 2
2	7	1	∵ 7 % 3 => 1
0	7	1	
1	8	2	
2	10	1	
0	10	1	
0	10	1	
1	11	2	
1	12	0	

State	Input 0	Input 1	Input 2	Output
$\Rightarrow$ A	A	B	C	0
B	B	C	A	1
C	C	A	B	2



This can be used in real

i.e

A	B	C
0	1	2
Equal	Large	Small

Happy Angry depressed

Sing Curse Sleep

	Input			
State	Homework	Party	Poor exam	output
A (Happy)	A	A	B	SING
B (Angry)	C	A	B	CURSE
C (depressed)	C	A	C	SLEEP