S-attribute

① Synthesized attribute.

② Semantic actions are ~~play~~ placed at the end of the right hand side of the production.

③ Evaluated bottom up parsing manner.

L-attribute

① synthesized / inherited. (If inherited, only parents and left siblings).

② Anywhere in right hand side.

③ Depth first left to right manner.

$A \rightarrow BCD$

$B \rightarrow C.val \rightarrow B.val$

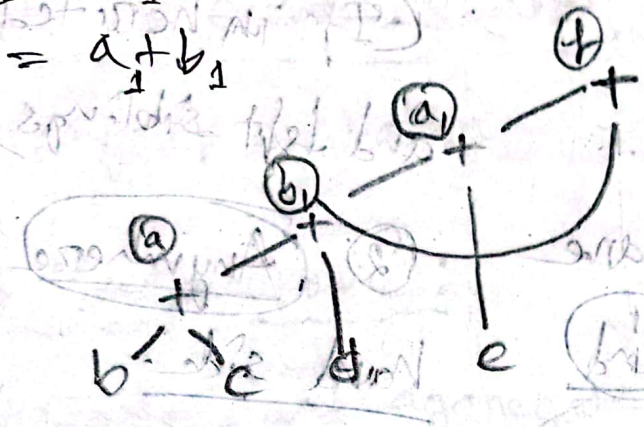
Pg For same operant, (right, left same operant)

$a = b + c$ ——

$b_1 = a + d$

$a_1 = b_1 + e$

$t = a_1 + b_1$



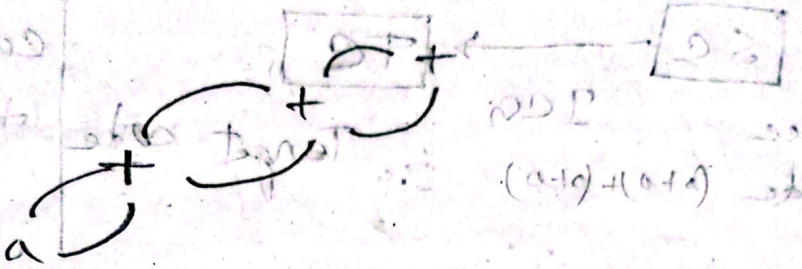
Benefits of Intermediate code generator:

1. Portability will be ~~enhanced~~ enhanced.
2. Retargeting is facilitated.
3. Machine Independent.

DAG

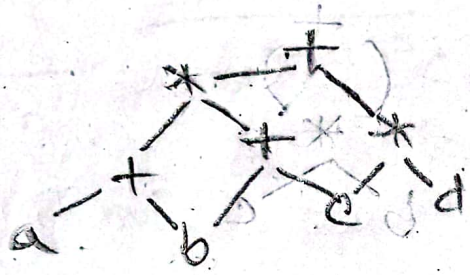
P1

((a+a)+(a+a))+((a+a)+(a+a))



P2

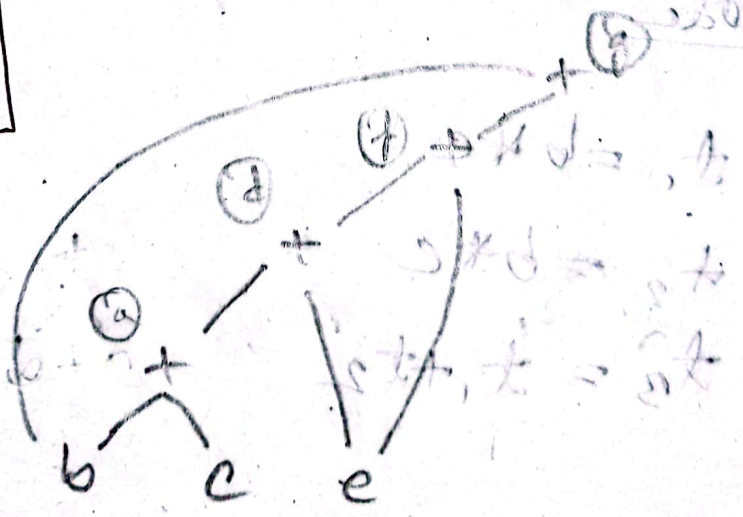
(a+b) * (b+c) + (c*d)



P3

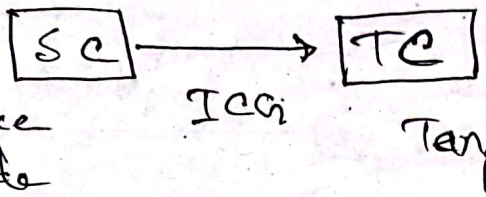
a = b+c
d = a+c
f = d+e
g = b+f

block of code



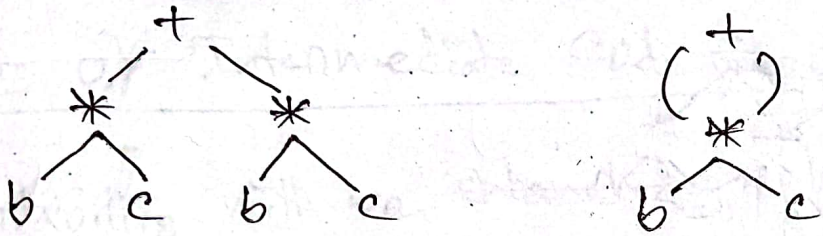
Intermediate Code Generation

Machine Independent



Compiler use intermediate code to generate the target code.

Example: (b * c) + (b * c)



Postfix → bc * bc * +

3 address code

$t_1 = b * c$

$t_2 = b * c$

$t_3 = t_1 + t_2$

Intermidiate Code generation Representation

1. 3 address code.
2. Syntax tree.
3. Postfix notations.
4. DAG

Semantic two rule

Synthesized

A → BCD

A.val → B.val

Inherited

A → BCD

~~B.val~~ → ~~B.val~~

B.val → A.val

Annotated Parse tree

SDD V.S SDT difference

SDD	SDT
Syntax directed Definition	Syntax directed translation
① CFG + Semantic rule.	① CFG + Semantic rules + Semantic action.
② Mostly used for 'Specification'.	② Implementation used.
③ Ease to use	③ Efficient to use.
④ Eval = Eval + Tval	④ { print ('+') }

⑥ Types of Semantic analysis errors:

① Types mismatch

② Accessing variable out of scope.

③ Declare variable multiple times within scope.

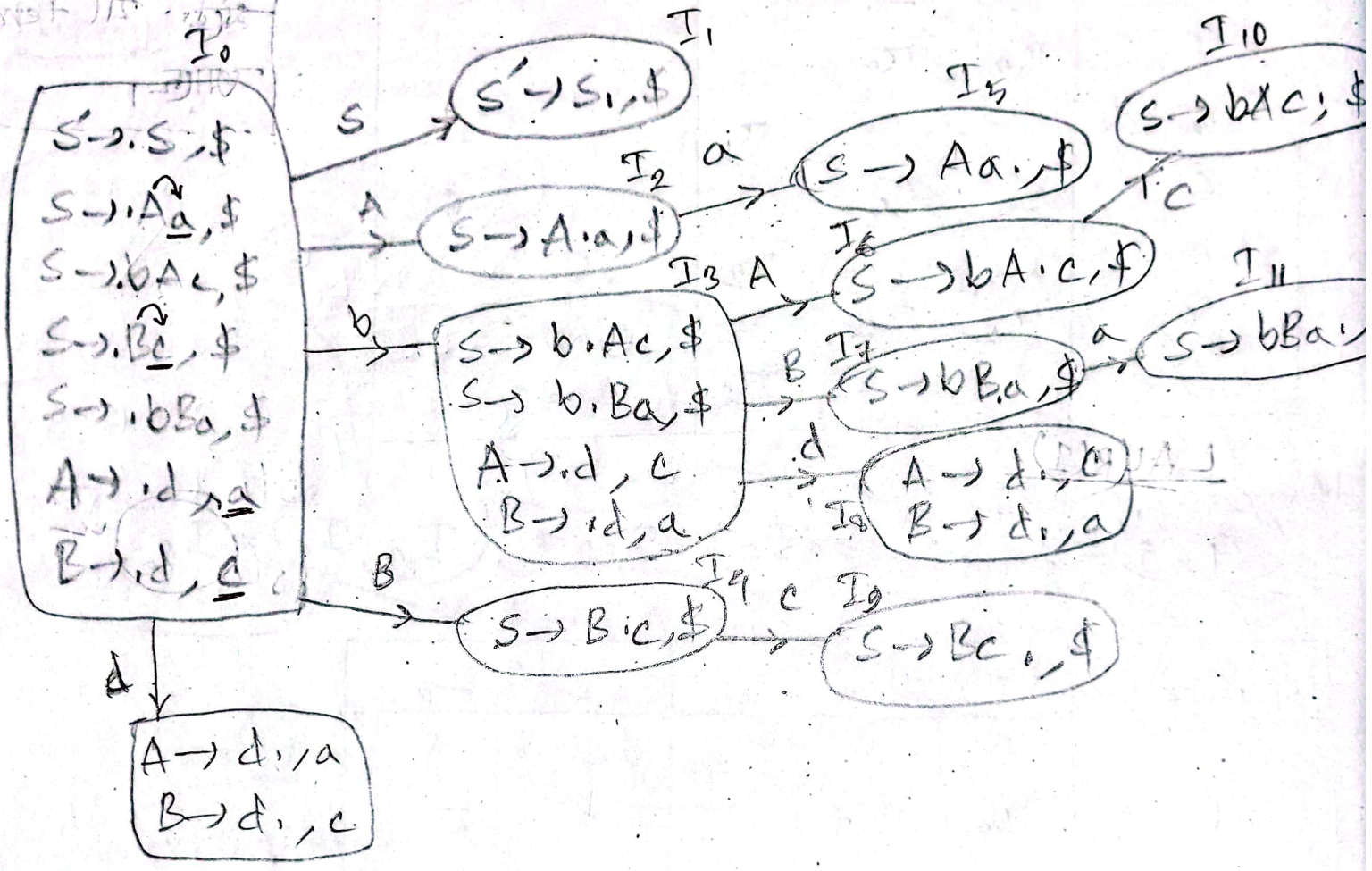
④ Reserve identifier misuse.

⑤ Undeclared variable.

⑥ Actual and formal parameter mismatch.

Practice CLR(1) and LALR(1)

	First	Follow
$S \to Aa \mid bAc \mid Bc \mid bBa$	d, b,	$
$A \to d$	d	a, c,
$B \to d$	d	c, a



steps
- DFA তৈরী
- lookup তৈরী করা,
- reduce production lookup এর (action) বসানো।

CLR(1)

State	Action a	Action b	Action $	GOTO S	GOTO A
0	S_3	S_4		1	2
1			accepted		
2	S_6	S_7			5
3	S_3	S_4			8
4	r_2	r_3			
5			r_1		
6	S_6	S_7			9
7			r_3		
8	r_2	r_2			
9			r_2		

reduce হওয়ার Lookup ahead এ, যেটা Lookup হলো action এ, সেটার terminal আছে।

LALR(1)

Merge $(4,7) = I_{47}$, $(I_8, I_9) = I_{89}$, $(I_3, I_6) = I_{36}$

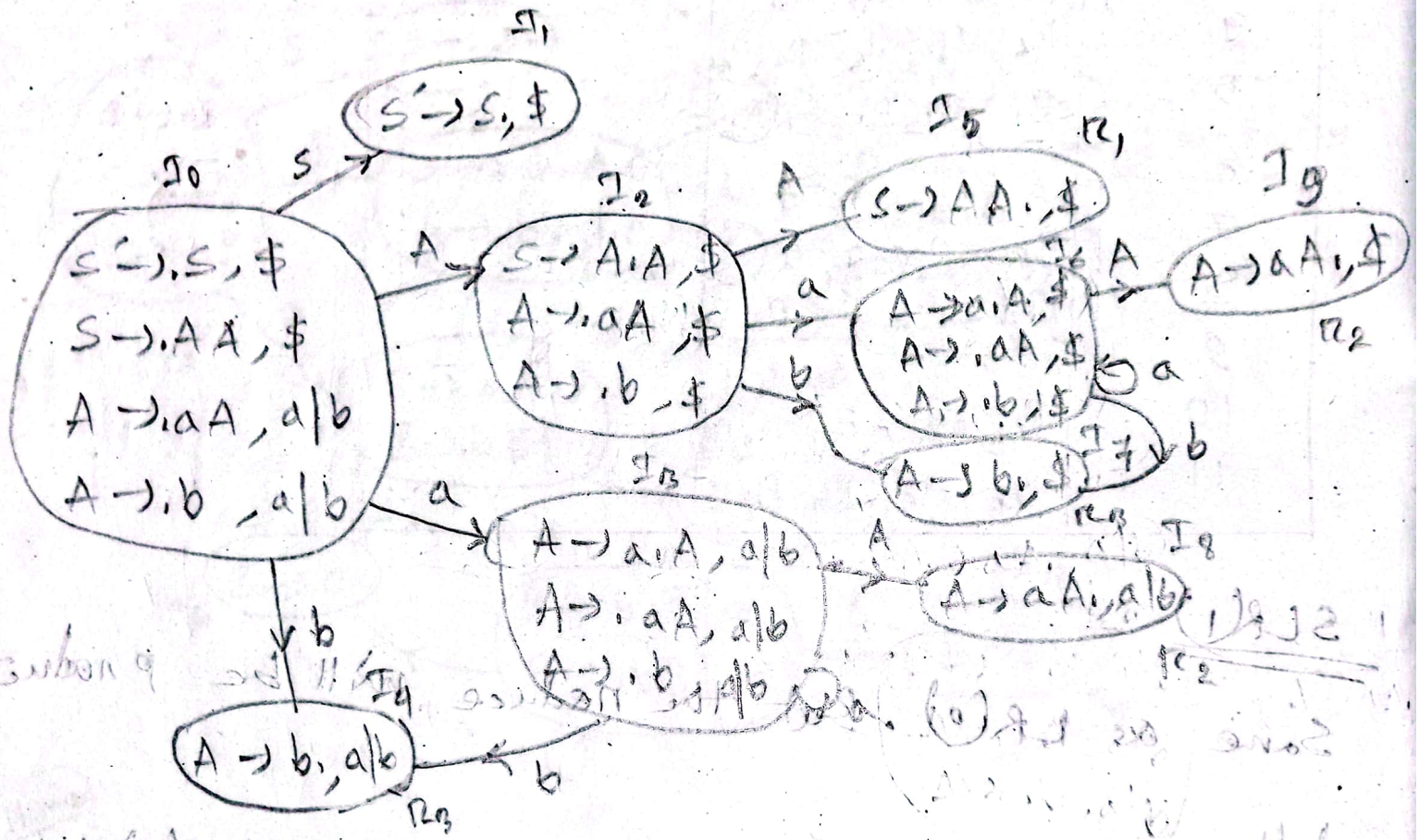
State	Action a	Action b	Action $	GOTO S	GOTO A
0	S_{36}	S_{47}		1	2
1					
2	S_{36}	S_{47}			5
36	S_{36}	S_{47}			89
47	r_3	r_3	r_3		
5			r_1		
89	r_2	r_2	r_2		

Problem CLR(1) and LALR(1)

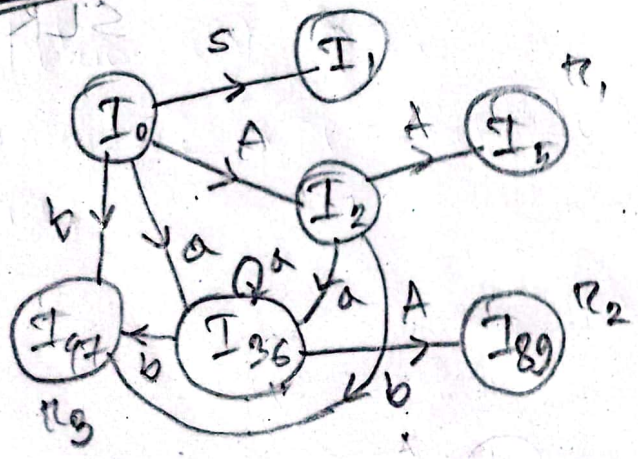
	first	follow
S → AA	a, b	$
A → aA (2) | b (3)	a, b	a, b

use 2 times

DFA



For LALR



LR(0)

state	Action				goto			first	follow
	d	a	b	c	$	A	B	S		
0	s_2	s_3						1	d, a	$
1					accepted				b, c	$
2			s_6	s_7		5			d, b, c	$
3			s_9	s_4			8			
4	r_2	r_2	r_2	r_2	r_2					
5	r_1	r_1	r_1	r_1	r_1					
6			s_6	s_7		10				
7	r_4	r_4	r_4	r_4	r_4					
8	r_2	r_2	r_2	r_2	r_2					
9				s_4			11			
10	r_3	r_3	r_3	r_3	r_3					
11	r_5	r_5	r_5	r_5	r_5					

SLR(1)

Same as LR(0). But the reduce will be production follow.

LR(0) ~
SLR(1) ~

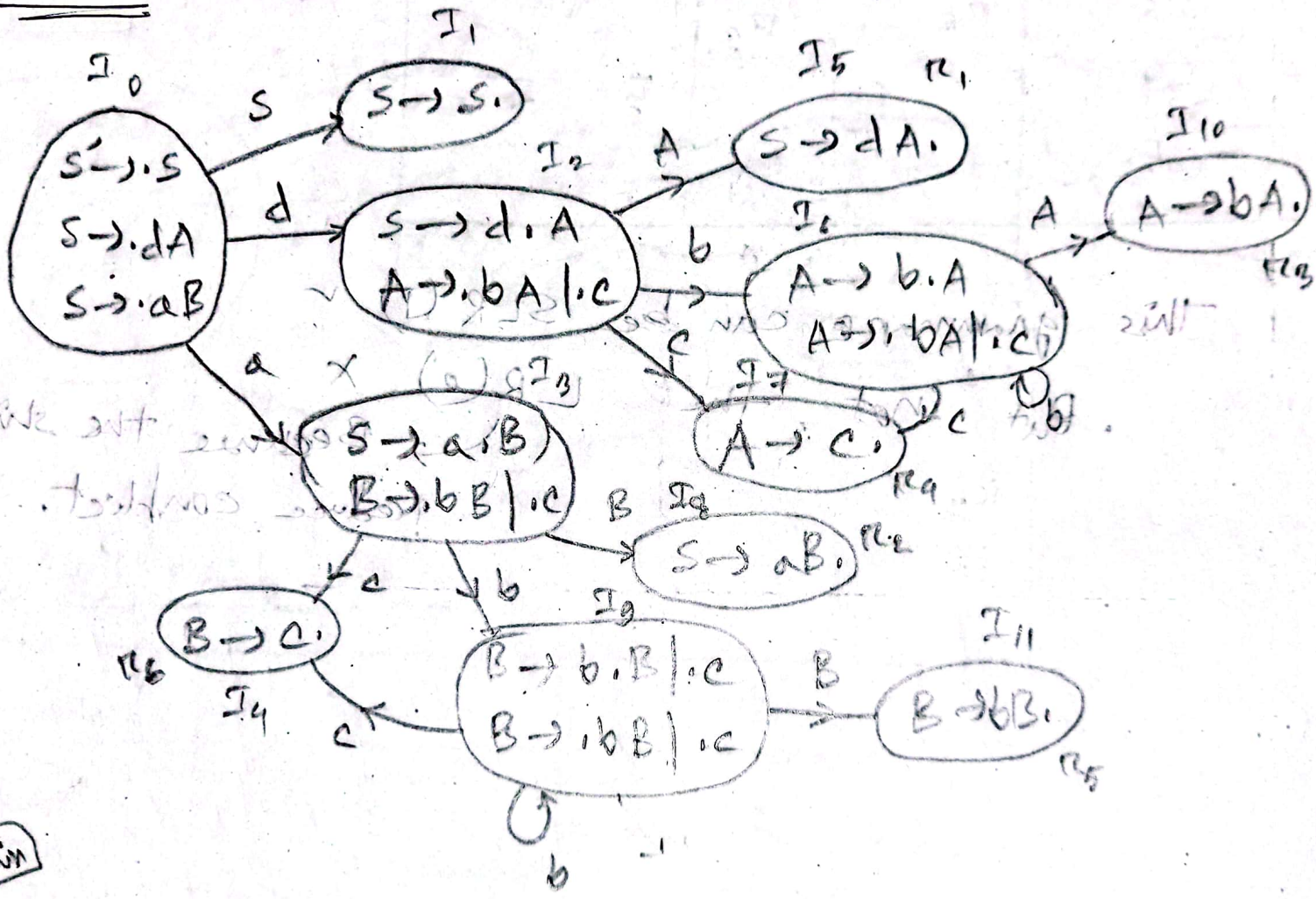
Problem 2 LR(0) and SLR(1)

$S \rightarrow dA_1 \mid aB_2$

$A \rightarrow bA_3 \mid c_4$

$B \rightarrow bB_5 \mid c_6$

DFA



1 min

SLR(1)

State	Action			S_{10} to	
	+	id	$	E	T
0		S_3		1	2
1			completed		
2	S_4		r_2		
3	r_3		r_3		
4		S_3		5	2
5			r_1		

Total 23 min

First	Follow
0	$ ✓
1	+, $ ✓

$r_1, r_2 \rightarrow \$$

$r_3 \rightarrow +, \$$

This grammar can be SLR(1) ✓

But not the LR(0) ✗

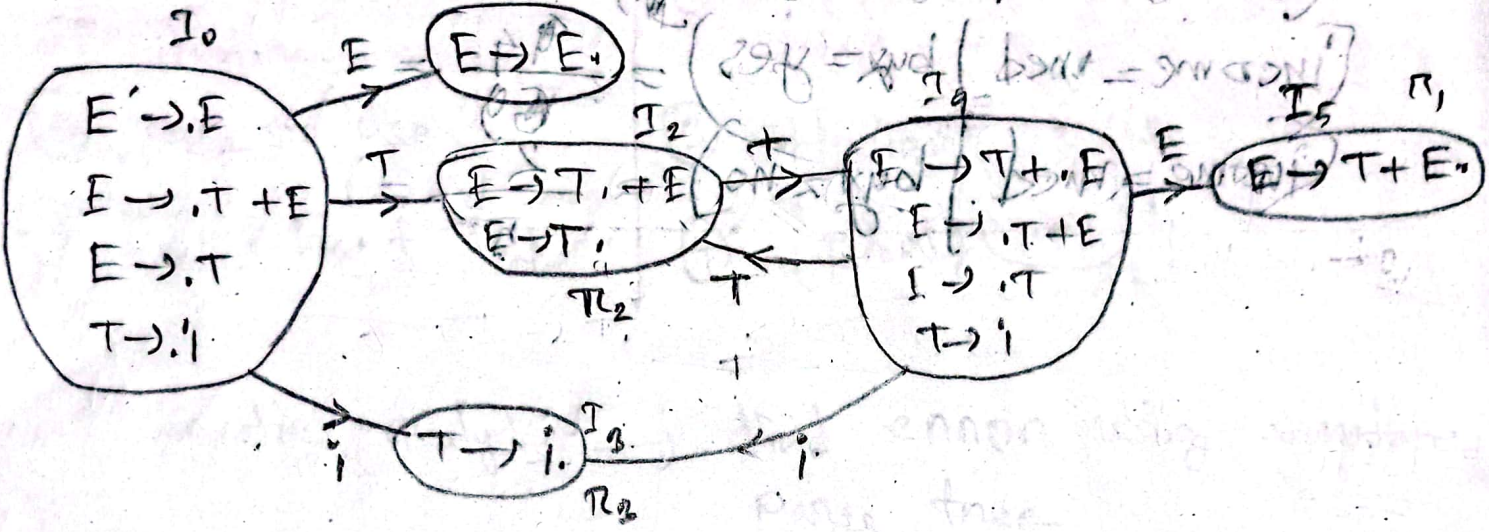
↳ Because the shift reduce conflict.

Problem LR(0) and SLR(1)

$E \to T + E \;(1) \mid T \;(2)$

$T \to i \;(3)$

DFA



LR(0) Parsing table:

State	Action			GO	
	+	i	$	E	T
0		S_3		1	2
1			complete		
2	S_4 / π_2	π_2	π_2		
3	π_3	π_3	π_3		
4		S_3		5	2
5	π_1	π_1	π_1		
6					

Annotated Parse tree

SDD	SDT
① CFG + Semantic rule	① CFG + Semantic rules + Semantic Action
② Mostly used for specification.	② Implementation
③ ease to use.	③ efficient to use.
④ Eval = Eval + Tval	④ {print("+")}

Semantic analysis → find error using annotated parse tree

LALR(1) CLR(1)

LR(1) item = (LR(0) item + lookahead)

S → .aA, a/b

LR(0), SLR(1), LALR(1) এর total state same হয়, কিন্তু CLR(1) এর তাদের তুলনায় বেশী হয়,

CLR(1) ≥ LR(0) = SLR(1) = LALR(1) → Same state

lalbr

lookup ahead

LR(0)

→ DFA

→ Dot এর আগে Production থাকলে তার Production লিখতে হয়,

→ reduce row তে কিছু দেয়া নেই,

CLR(1)

→ DFA

→ table এর reduce (r1, r2 ..) এর lookup এর action এ,

SLR(1)

→ DFA

→ table এ reduce (r1, r2 ..) production এর আগে Follow এ বসাতে,

E → ab

E এর follow এ (r1 ..) বসাতে

LALR(1)

→ DFA

→ Merge করা

→ table এ Merge state এর নাম একটা,

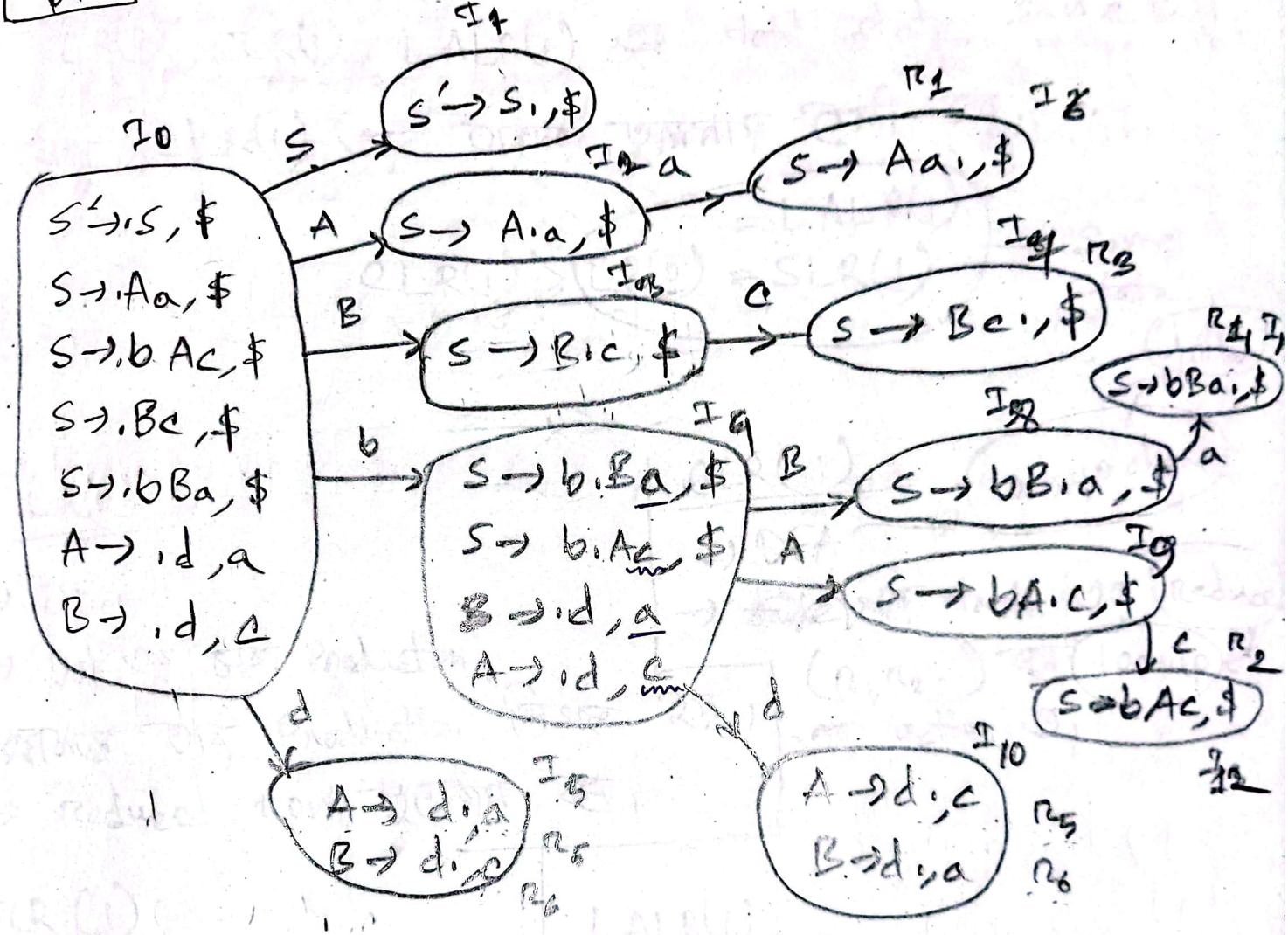
$S \to Aa \mid bAc \mid Bc \mid bBa$

$A \to d$

$B \to d$

CLR, LALR

DFA



State	Action					GOTO		
	a	b	c	d	$	S	A	B
0		s_4		s_5		1	2	3
2	s_6							
3			s_7					
4/5	r_5		r_6	s_{10}			9	8
6	s_{11}							
9			s_{12}					
10	r_6		r_5					
6					r_1			
12					r_2			
11					r_4			

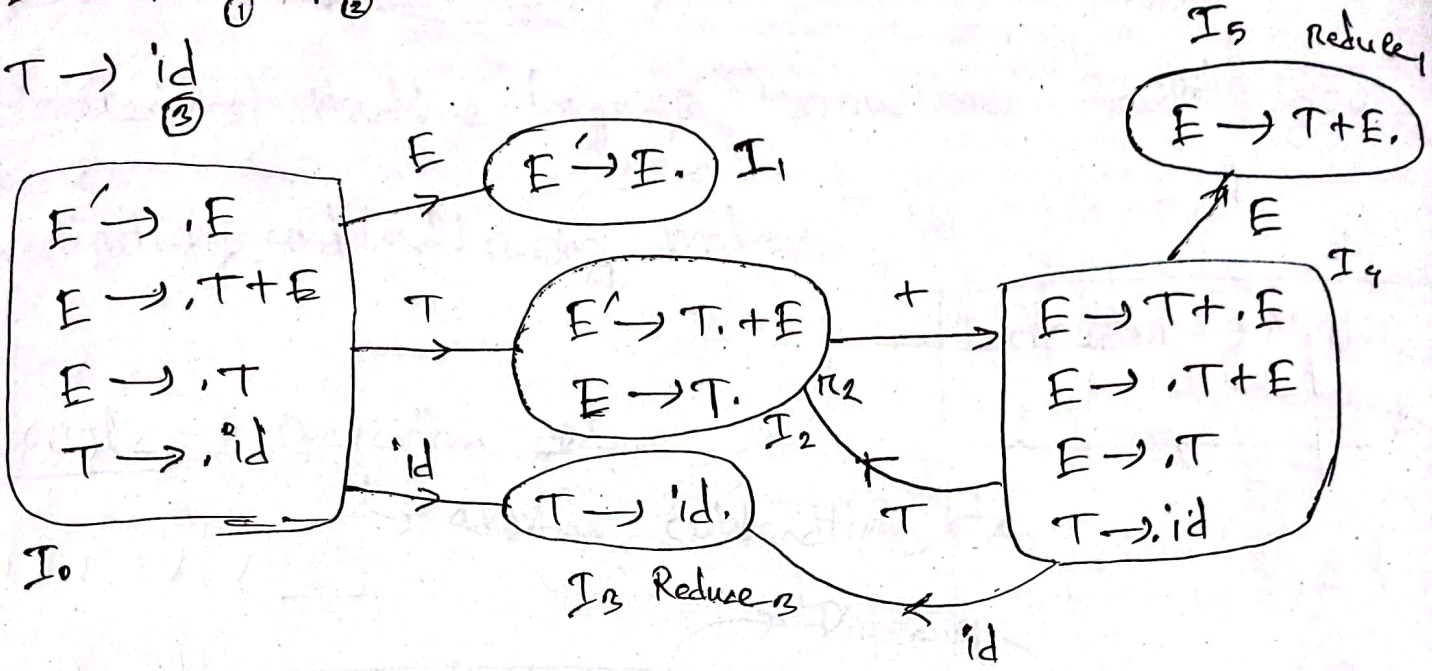
5 —

LR(0) Parsing table ← (Bottom up)

LR(0) CI (Canonical Items)

$E \to T + E \;(1) \mid T \;(2)$

$T \to id \;(3)$



যদি (.) এর পর যদি Non terminal হয় তাহলে তার Production এর সমান হবে।

state	'id	+	$	E	T
0	S_3			1	2
1	~~r_2~~	~~S_4/r_2~~	~~r_2~~		
2	r_2	S_4/r_2	r_2		
3	r_3	r_3	r_3		
4	S_3			5	2
5	r_1	r_1	r_1		

LR(0) ✗ (Because shift-reduce conflict)

Prob 1

	Compiler First	Follow
$E \to TE'$	[, <, >, a, b	], $
$E' \to ?E \mid \epsilon$	?, ϵ	], $
$T \to FT'$	a, b, <, >, [	?,], $
$T' \to !T \mid \epsilon$	!, ϵ	?,], $
$F \to WF'$	a, b, <, >, [	!, ?,], $
$F' \to @F \mid \epsilon$	@, ϵ	!, ?,], $
$W \to uv \mid v \mid [E]$	a, b, <, >, [	@, !, ?,], $
$v \to a \mid b$	a, b	@, !, ?,], $
$u \to < \mid >$	<, >	a, b

Case 1

যদি কোন non terminal শুধু terminal পায় তাহলে non-terminal এর First হবে।

Case 2

যদি কোন কিছু না পায়, তাহলে Parent এর Follower বসবে।

ϵ-closer(A) = {A, D, B}

ϵ- " (B) = {B}

ϵ- " (C) = {C, D, B}

" " (D) = {B, D}

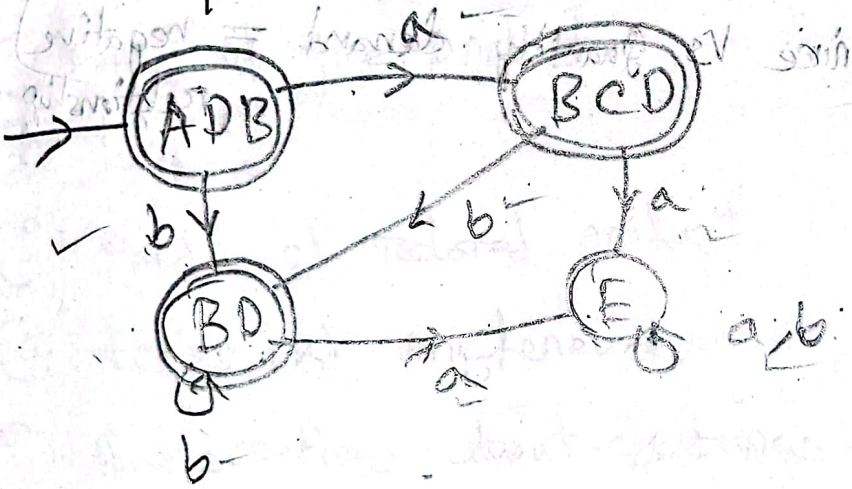
NFA

	a	b
A	BC	ϕ
B	ϕ	D
C	ϕ	ϕ
D	ϕ	ϕ

DFA

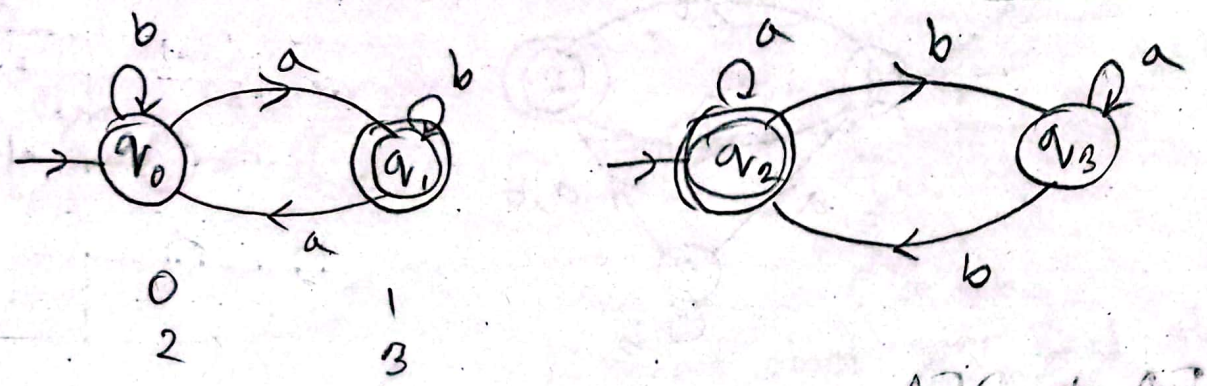
	a	b
ADB	BCD	BD
BCD	E	BD
BD	E	BD
E	E	E

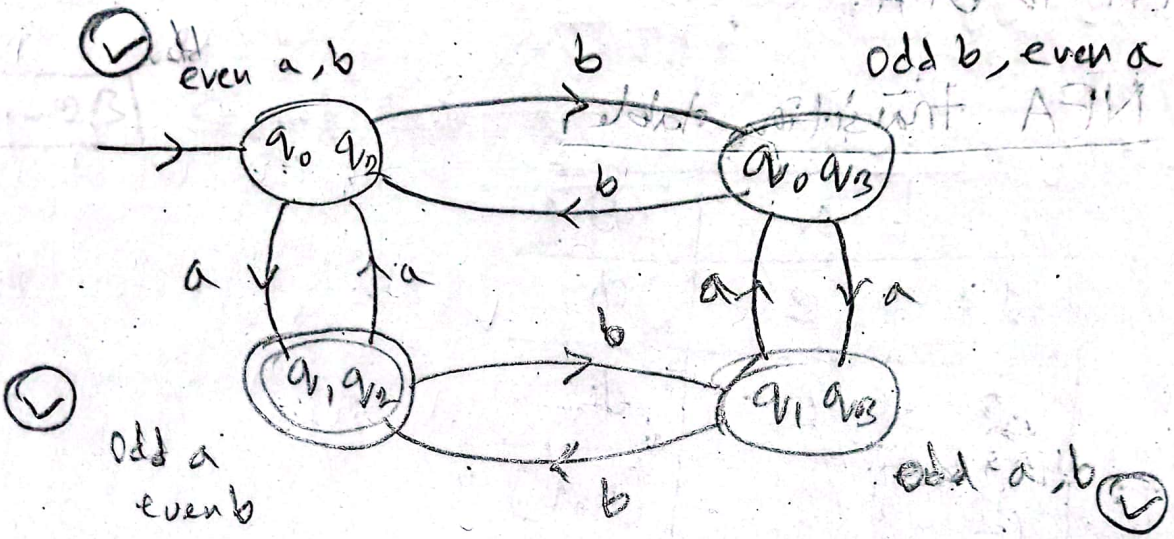
ϵ এর value আসে

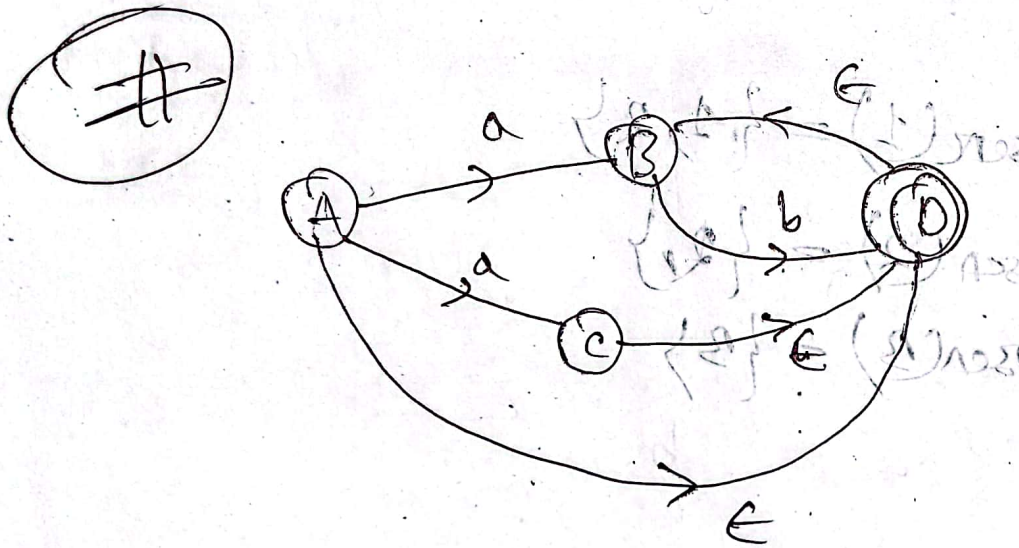


ϵ এর ক্ষেত্রে, DFA এর state এর [ADB] প্রতিটির যান NFA এর state গুলোর হবে, Then তার প্রত্যেকটি value জন্য আলাদা ϵ এর value গুলো DFA র মধ্যে।

Odd 'a' and even b.

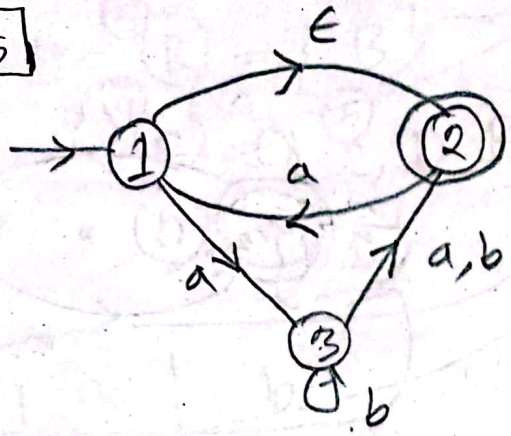






NFA to DFA (ϵ)

Problem-25



NFA to DFA.

Sol NFA transition table:

	a	b
1	3	ϕ
2	1	ϕ
3	2	{2,3}

Here,

ϵ-closer(1) = {1, 2}

ϵ-closer(2) = {2}

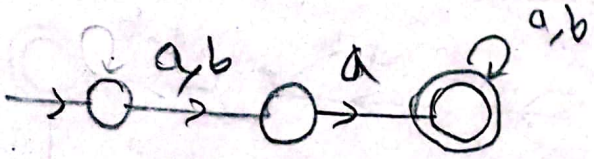
ϵ-closer(3) = {3}

DFA table:

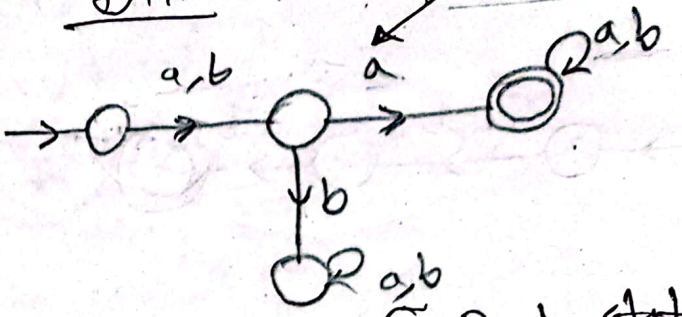
	a	b

Problem-22 Second a start,

NFA



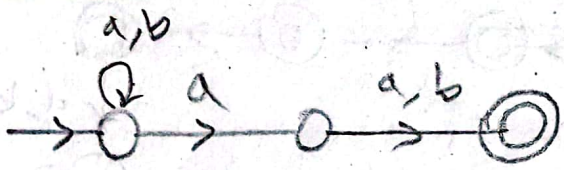
DFA



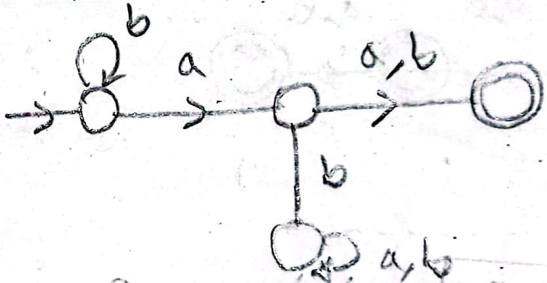
∴ Second start ~~of~~ Dead state

Problem-23 Second last a,

NFA

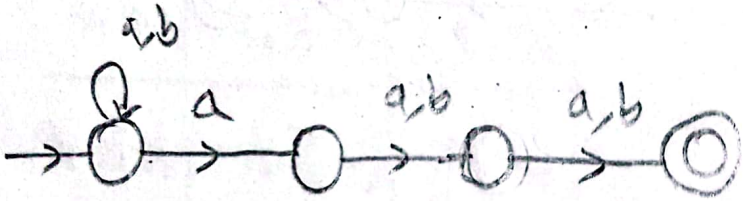


DFA

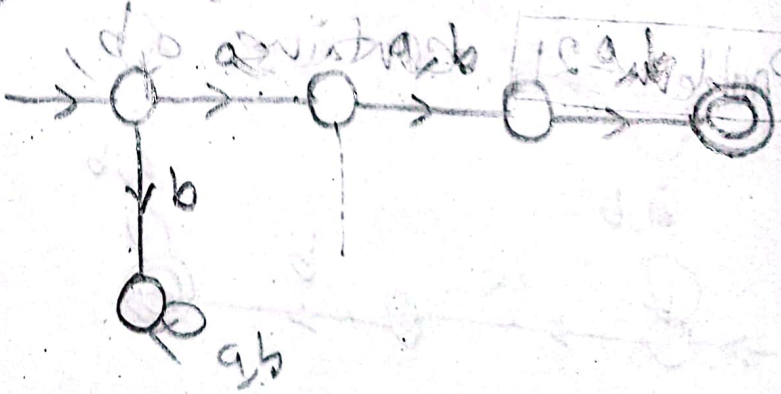


Problem-24 Third last a,

NFA



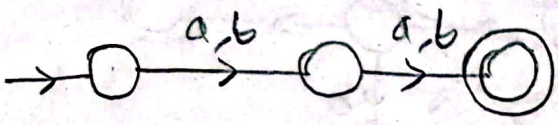
DFA



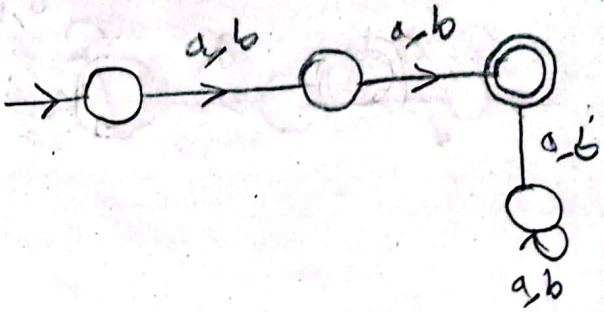
NFA Practise

Problem 18 exactly 2,

NFA

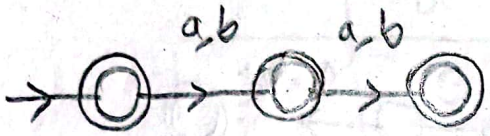


DFA

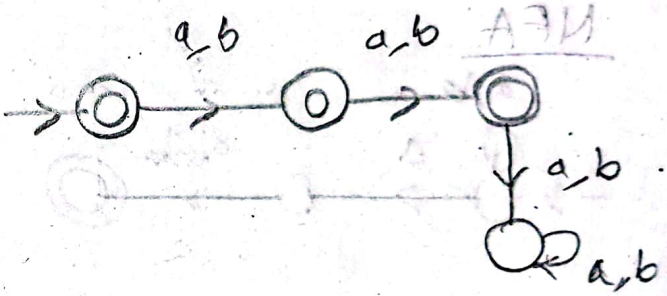


Problem 19 Maximum 2,

NFA



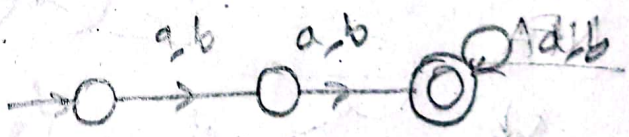
DFA



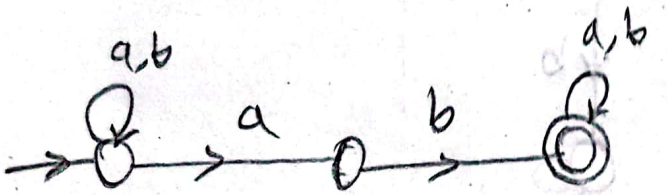
Problem-20 minimum 2,

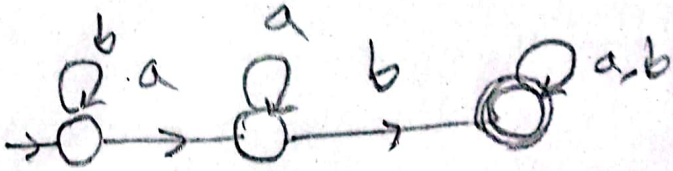
NFA

DFA

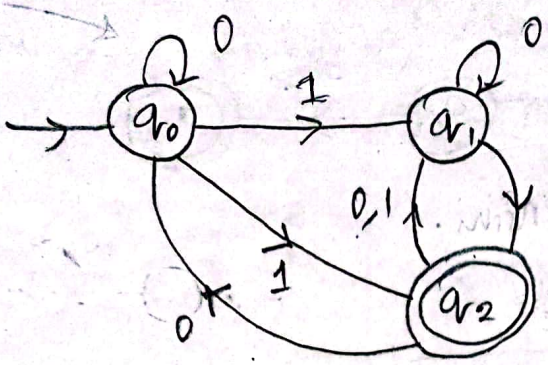


Problem-21 contains ab,





Problem 17



NFA to DFA conversion.

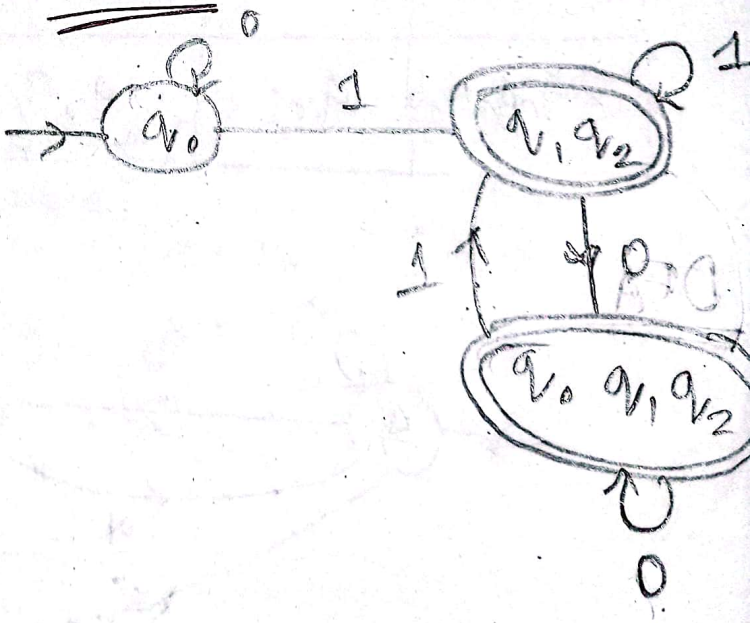
NFA transition table:

	0	1
q_0	$q_0 q_2$	$q_1 q_2$
q_1	$q_1 q_2$	q_2
q_2	$q_0 q_1$	q_1

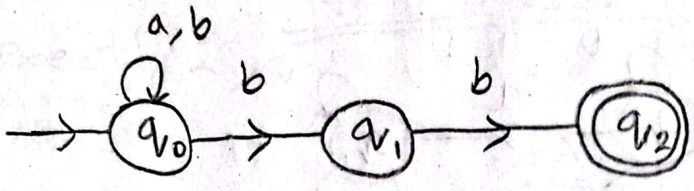
DFA Transition table:

	0	1
q_0	q_0	$q_1 q_2$
$q_1 q_2$	$q_0 q_1 q_2$	$q_1 q_2$
$q_0 q_1 q_2$	$q_0 q_1 q_2$	$q_1 q_2$

DFA:



Problem-16



← NFA

NFA to DFA conversion.

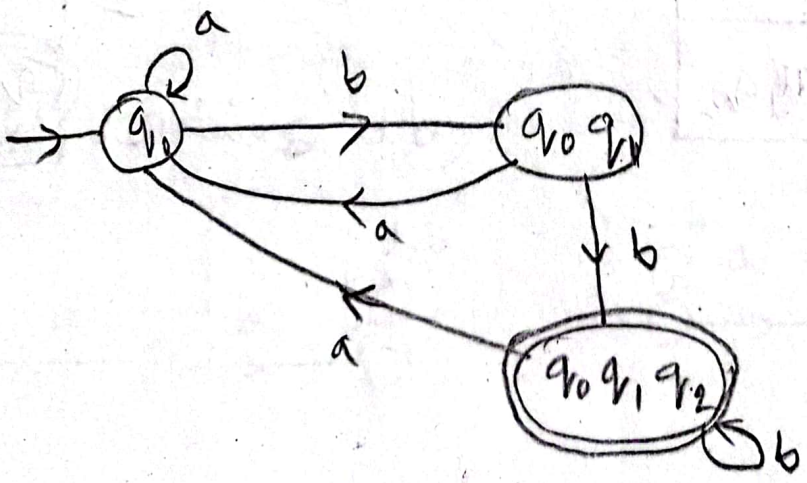
NFA transition table:

	a	b
q_0	q_0	q_0, q_1
q_1	ϕ	q_2
q_2	ϕ	ϕ

DFA table:

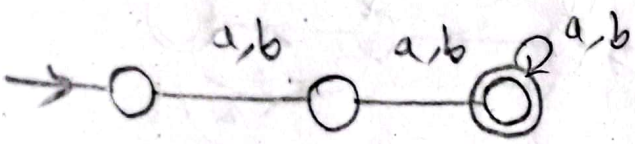
	a	b
q_0	q_0	$\{q_0 q_1\}$
$\{q_0 q_1\}$	q_0	$\{q_0 q_1 q_2\}$
$\{q_0 q_1 q_2\}$	q_0	$\{q_0 q_1 q_2\}$

DFA

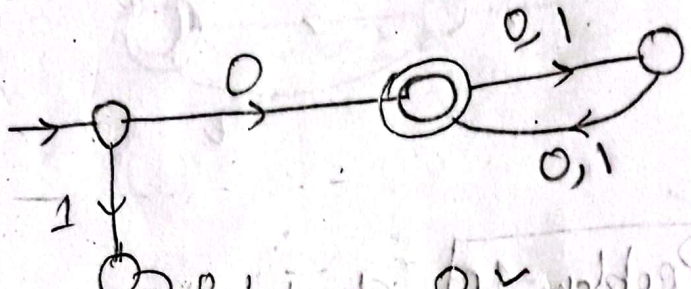


q_2 final state

Problem 12 minimum length 2.

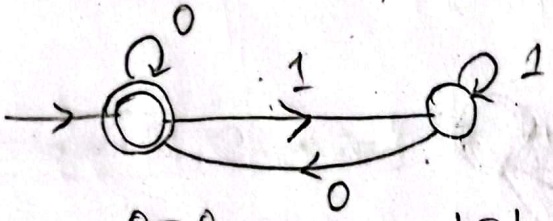


Problem-13 Start with 0 & has odd length.



0
01 x
00 x
010

Problem-14 Divisible by 2.

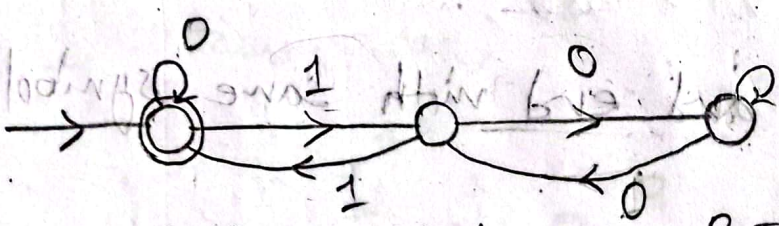


0, 1, 2
↓ ↓ ↓
0 1 10

0=0
2=10
4=100

1=1
3=11
5=101

Problem-15 Divisible by 3.

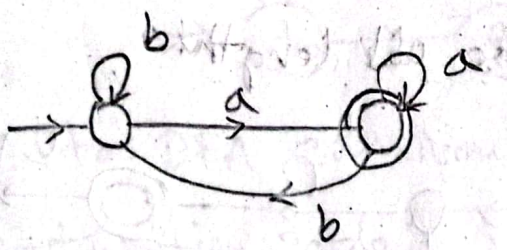


0=0
3=11

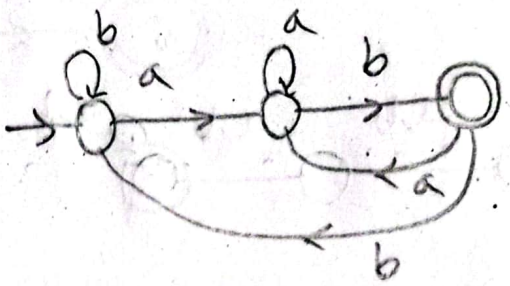
1=1
4=100

2=10
5=101

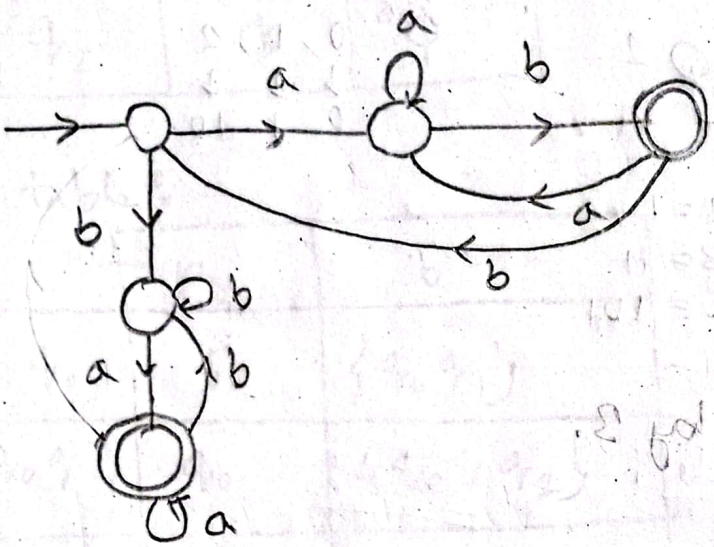
Problem-8 end with a,



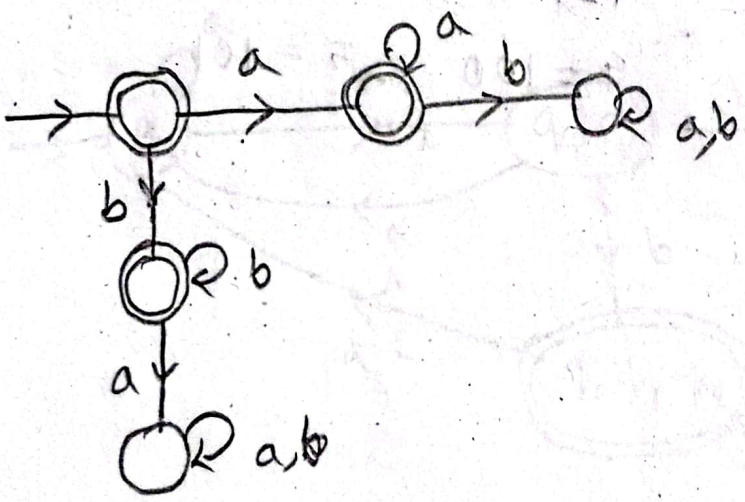
Problem-9 end with ab,



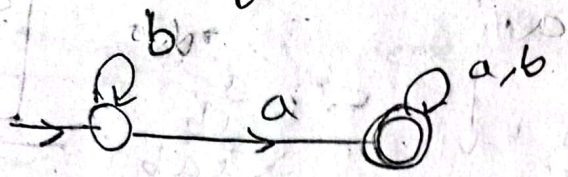
Problem-10 start with 'a' and 'b' or start 'b' and ends with 'a',



Problem-11 start and end with same symbol.

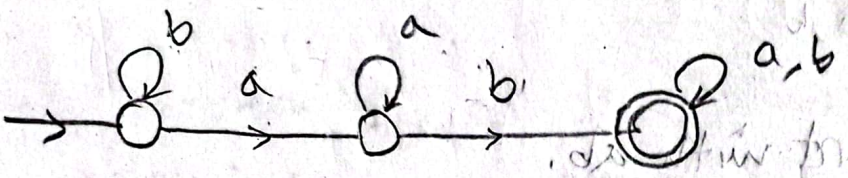


Problem-4 String contains a,



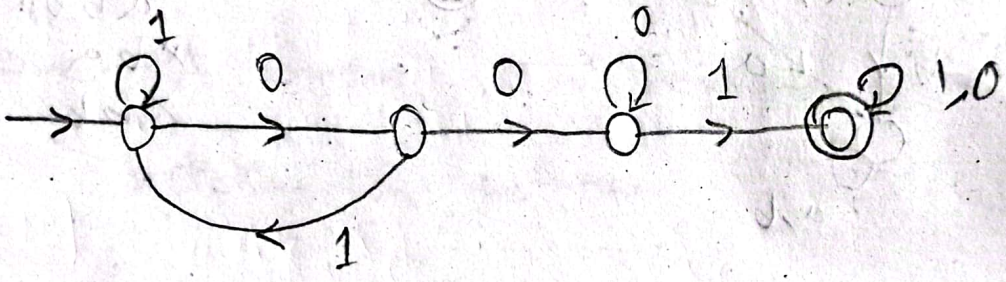
- baa
- bab
- bba

Problem-5 contains a,b,



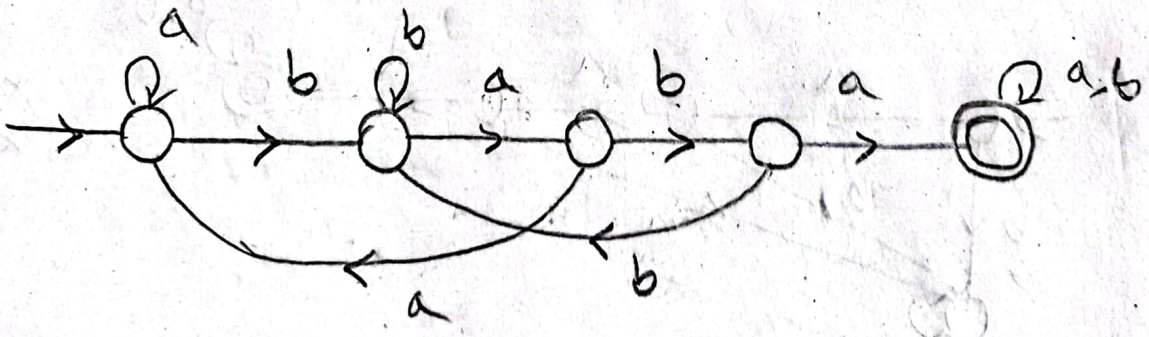
- bababa
- babab
- babba
- bbaab

Problem-6 contains 001,



- 10010
- 10011
- 1101
- 0001
- 1011

Problem-7 contains baba,

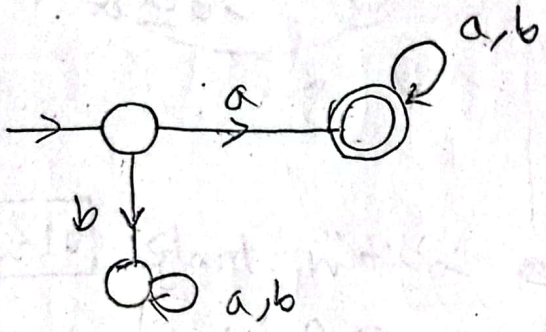


- bbaaba
- bbaaa
- babbaba

DFA

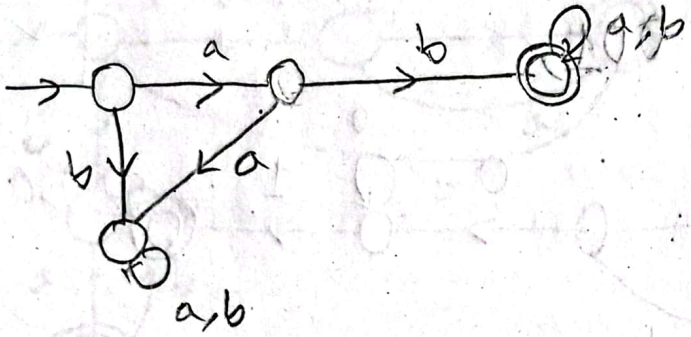
Example-1 start with a.

Start with + 2 = total State



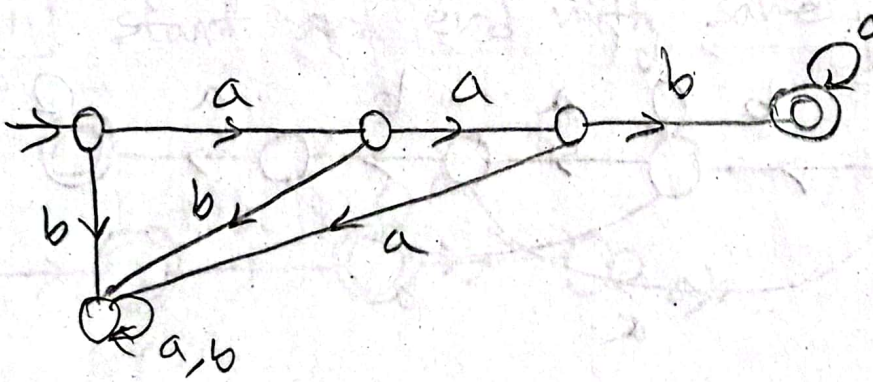
a
ab
aab
abb
~~x~~ baa

Example-2 Start with ab.



ab aab
~~x~~ aa
abba
~~x~~ ba
~~x~~ bb

Example-3 Start with aab,



aabaab
~~x~~ abb
~~x~~ aaa

String → Collection of alphabet.

'w' দ্বারা প্রকাশ করা হয়।

যেমন: ab, aab, babb.

$|w| = 7$, $w = ababbab$

(String এ total symbol সংখ্যা)

Language → Collection of strings over a particular rule. 'L' দ্বারা প্রকাশ করা হয়।

$L = \{aa, ab, ba, bb\}$ ← (যদি ৩ না হয় string length 2)

যদি ৩ না হয় max 2 হওয়া হবে,

$L = \{a, b, aa, ab, bb, ba, \in\}$

(খালি সেট না)

Compiler → Higher level language ক assembly code ↓ convert করে।

Assembler → Assembly language ক object file convert করে,

Linker → Build in function এবং library ব্যবহার করে link করে

Loader → Execute file create করে (.exe) দেয়;

(.exe)

Theory of computation:

Symbol → Building block / smallest element of a language.

Alphabet → Finite collections of symbols.

'Σ' দ্বারা প্রকাশ করা হয়,

উদাহরণ ↓

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

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

$$\Sigma = \{0, 1, \ldots 9\}$$

Compiler

Compiler এর 6 stage. যাতায় প্রথম High level language কে assembly language এ convert করা,

1) lexical analysis.
2) Syntax analysis.
3) Semantic analysis
4) Intermediate code generator.
5) Code optimization.
6) Code generation

Language processing System এ 5টি phase থাকে,

(1) Hypertext preprocessor → File inclusion (#include header file remove করা)
→ Macro expansion (constant remove করা)
(2) Compiler
(3) Assembler
(4) Linker
(5) Loader.