Date 23/July.
Page

Compiler construction

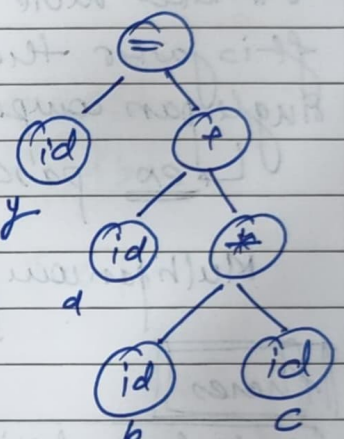
front end phase
similar for all machine.

Phases (6).

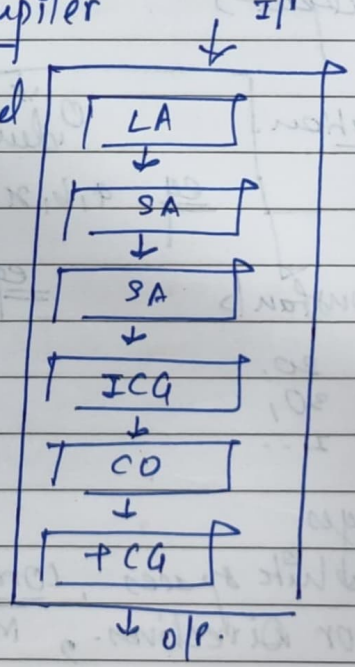
1. Lexical Analysis (tokens)
2. Syntax Analysis
3. Semantic Analysis
4. Intermediate code generation (3 address code)
5. code optimization
6. code generation / target-machine dependent

Assembly language code
MUL R_2, R_3
ADD R_1, R_2
store y, R_1

ex $y = a + b \times c$
$id = id + id \times id$



One pan compiler



1. fntend is high level language
2. output - target language.

1. Creation of one pan compiler faster
2. Memory extra.
3. whole are considered one.

fu 2 pan

1. first 4 is one pan
2. & last 2 are another pan.

- fu an implementation of compiler combination or one or more phases combined called pan fu grouping depend on 2 factors structure of source language
  2. Environment in which compiler operates (platform).

Sangam

Date
Page

① one pan compiler which panes through the source code of each compilation unit only.
② The whole compiler is kept in a main memory so it takes more space.
③ It is faster than multipan compiler
④ Single pan compiler no code optimization is used.
↳ ex pascal compiler.

Multipan compiler - C, C++.

Phases

① Lexical Analysis

↳ Stream of character (Input)
Tokens (output)

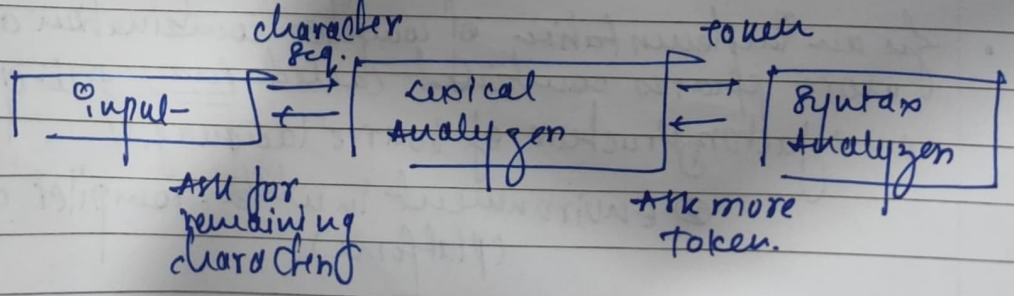
Step ① Tokenization.

Tokens

- Identifier — eg a, b, x
- keywords — eg. if, while, else, for
- Special symbols — eg @, $, #
- Operator — eg <, >, =, ==
- constants — eg 20, 30, 1...
- Operator — eg {, }, ;, (,), ...

Step ② Error messages

Step ③ Removes white spaces, comments, preprocessor directives, Macro



Sangam

Example:

```
int main (int a, int b)
{ //return min of two
  if (a<b)
    { return (a);
  else
    return b;
}
```

int - keywords
main - Page
{ → seperator
a → identifier

total tokens - 28

Example -

```
printf("i = %d, &i = %x", i, &i);
```

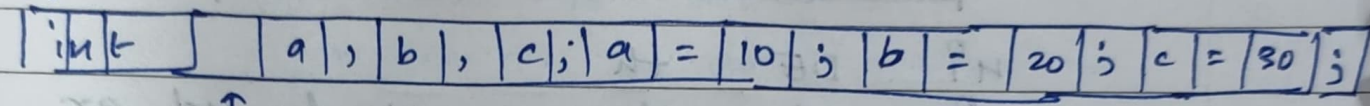
↳ count - Number of tokens. ⇒ 10

Sangam

* Input Buffer

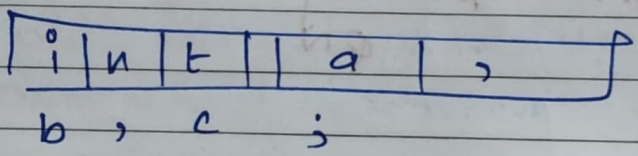
int a, b, c;
a = 10; c = 30;
b = 20;

bp — BP does not change position
fp — FP can change position.



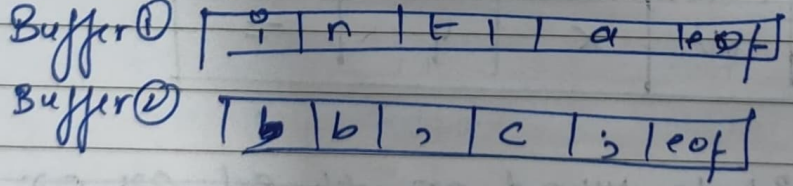
2 Schemes —— ① One Buffer
② 2 Buffer

① One Buffer



Now in this it overides the buffer this is the major drawback of one Buffer.

② 2 Buffer.



EOF - end of Buffer

There are now 2 buffer if the sentence is greater than the 2 buffer will get use alternatively.

(xx) Difference b/w multipass compile & one pass compiler

Lexical Analyser — Also called as scanner
└ Language — Lex

Sangam

(6) compiler construction tool kit - It provides an integrated set of Routines for constructing various phases of a compiler.

compiler construction tools.

→ The compiler writers like software developer uses modern development environment tools.

→ In addition to this generalized software development tools other more specialized tools have been created to implement various phases of compiler

→ This tools uses specialized language & algorithm for implementing specific components.

→ commonly used tools are — ① Parser generator

② Scanner generator

③ Syntax Directed translation engine

④ code generator generator

⑤ Data flow Analysis engine

① Parser generator – Automatically produce syntax Analyzer from a grammatical discription of a programming language.

② Scanner generator – It produce lexical Analyzers from a regular expression description of the token of a language.

③ Syntax Directed Traslation Engine – It produce collections of Routines for walking a pars spars tree & generating intermediate ~~code~~ code It ~~produce~~

④ code generator generator – It produce a code generator from a collection of Rules for translating each operation of the intermediate language into the machine language for a target machine

⑤ Data flow Analysis engine – It facilitate the gathering of info about how values are transmitted from one part of a program to each other part.
It is a key part of code optimization.

Sangam

Example

```
main()
{
  a = b++ + ++b - --b - --b + ++b + ++b == ;
  printf("%d %d", a, b);
}
```

↳ 30 ↳ 25

Example (28).

```
main()
{
  int a = 10;
  char b = "a b c";
  int c = 30;
  int /* comment */ *p = &m = 20;
}
```

* commenting data delete hai

UNIT-2

Q1 Role of Syntax Analyser.

→ Syntax Analysis also called as parsing is the second phase after lexical Analysis. It checks the syntactical structure of a given input. And it does so by building a datastructure called parse tree or syntax tree

The parse tree is constructed by using predefined grammer of the language & input-string.

Sangam

Production rule $\Rightarrow \alpha \rightarrow \beta$

(variable) $(V \cup T)^*$

* If the given input string can be produced with the help of syntax string then that input string said to be incorrect syntax.
* If not the error is reported by syntax Analyser.
* The main goal of syntax analysis is to create parse tree (Abstract syntax tree) (AST) of the source code which is Hierarchical Representation of the source code that reflect grammatical structure of the program.

Grammar – It is a finite set of formal rules for generating syntatically correct sentences or meaningful sentences.

① CFG [context free grammar]
works on [V, T, P, S]

V → (finite set of Non terminals) variables
T → —— " —— terminals
P → —— " —— production Rules
S → Start Symbol.

Types of grammar

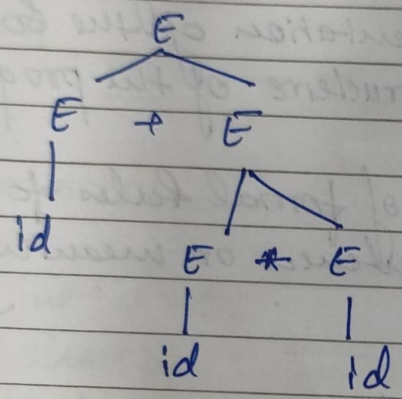
on Basis of production rule.	on basis of derivation tree	on basis of string
→ type 0	↳ Ambiguous	↳ Recursive
→ type 1	↳ Unambiguous	↳ Non-Recursive
→ type 2		
→ type 3		

Ambigous grammar

① $E \rightarrow E+E \mid E*E \mid id$

$\rightarrow id+E*E$

$\rightarrow id+id*id$



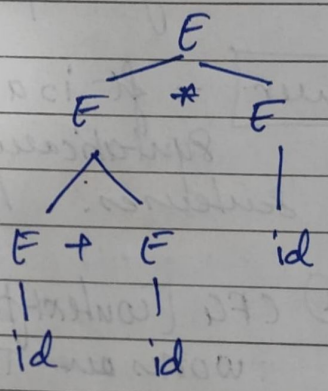
id + id + id

id + id * id

② $E \rightarrow E*E$

$\rightarrow E+E*E$

$\rightarrow id+id*id$



Example check wheater given grammar is ambigious or not-

① $S \rightarrow aS \mid Sa \mid a$

(Result) $w = aa$

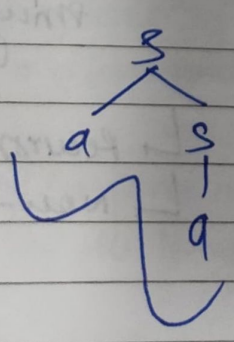
aS	Sa	a
aaa	aaa	a

~~$S \rightarrow a$~~

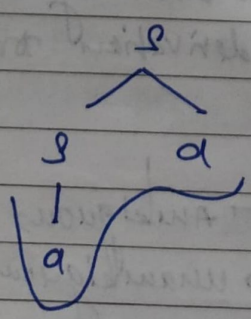
$S \rightarrow aS$

$\rightarrow aa$

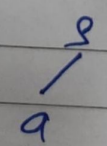
$S \rightarrow Sa$



$w = aa$



$w = aa$

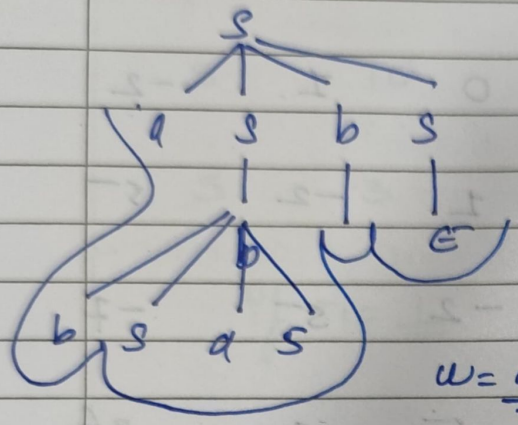


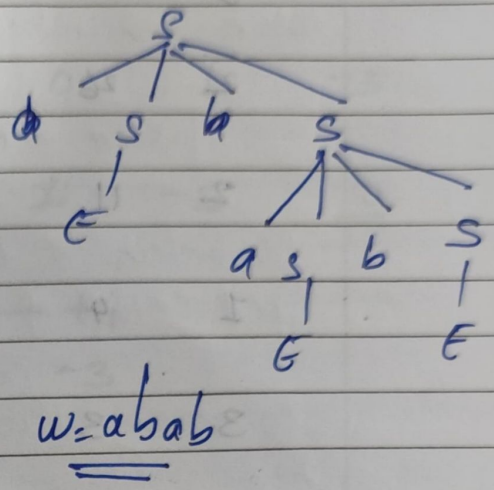
Ambigious grammar

Sangam

Date
Page

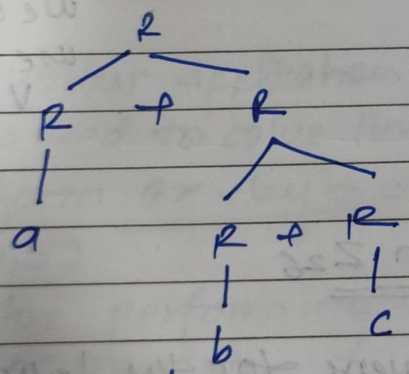
② $S \rightarrow aSbS \mid bSaS \mid \epsilon$ $\quad w = abab$

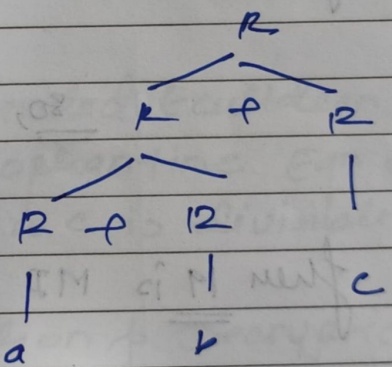




③ $R \rightarrow R+R \mid RR \mid R^* \mid a \mid b \mid c$ $\quad w \rightarrow a+b+c$

$R \rightarrow R+R$





CC

Date 30/July/2

Page

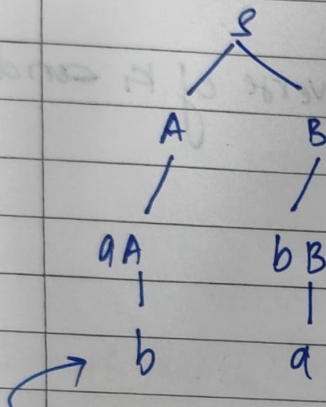
Ques ①

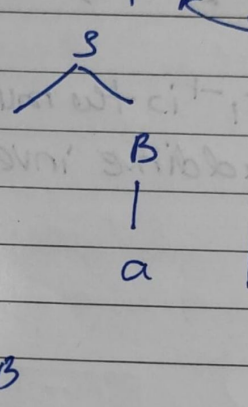
$S \to AB$

$A \to aA \mid b$

$B \to bB \mid a$

$w = abba$

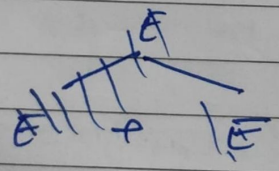




abba (unambigious)

Ques ②. $E \to E+E \mid E*E \mid id$ (Ambigious)

$w = id.$



$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to id.$

(converted to UNAmbigious)

WIOP

First & Follow

① First (A): It contains all terminals present in 1st place of every string denoted by A.

Example ① $S \to ABC \mid ghi \mid jkl$

$A \to a \mid b \mid c$

$B \to b$

$D \to d$

First(D) = {d}
First(B) = {b}
First(A) = {a, b, c}
First(S) = First(A,B,C) ∪ First(g h i) ∪ First(j k l)
= First(A) ∪ First(g) ∪ (j)
= {a, b, c, g, j}

(2) S → abc | xyz | ,c)

First(S) ⇒ {a, x, ,}

(3) S → ABC
A → a | b | ∈
B → c | d | ∈
C → e | f | ∈

First(C) = {e, f, ∈}
First(B) = {c, d, ∈}
First(A) = {a, b, ∈}
First(S) = {ABC}
= First(A) ∪ First(B) ∪ First(C)
= {a, b, c, d, e, f, ∈}

(4) S → AaAb | BbBa
A → ∈
B → ∈

First(B) → ∈
First(A) → ∈
First(S) → First(AaAb) ∪ First(BbBa)
(∈) (∈)
= {a, b}

Date
Page

(2)

$E \rightarrow +E'$

$E' \rightarrow *TE' / \epsilon$

$T \rightarrow FT'$

$T' \rightarrow \epsilon / +FT'$

$F \rightarrow id / (E)$

First(F) = {id, (}

First(T') = {ε, +}

First(T) = {F} ⇒ {id, (}

First(E') = {*, ε} ⇒

First(E) = {T} ⇒ {id, (}

Follow(A) – It contain set of all terminals which appear Immediately Right to the Non-terminal.

Example

$S \rightarrow ACD$

$C \rightarrow a / b$

∴ C is non terminal here

Follow(S) = {$}

Follow(A) = First(C)
= {a, b}

Follow(C) = First(D)
= {$}

Follow(D) = Follow(S)
= {$}

For last symbol you need to follow starting / first

Sangam

Date
Page

Example S - Aa Ab | Bb Ba

A → ε

B → ε

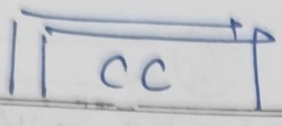
Follow(S) = {$}

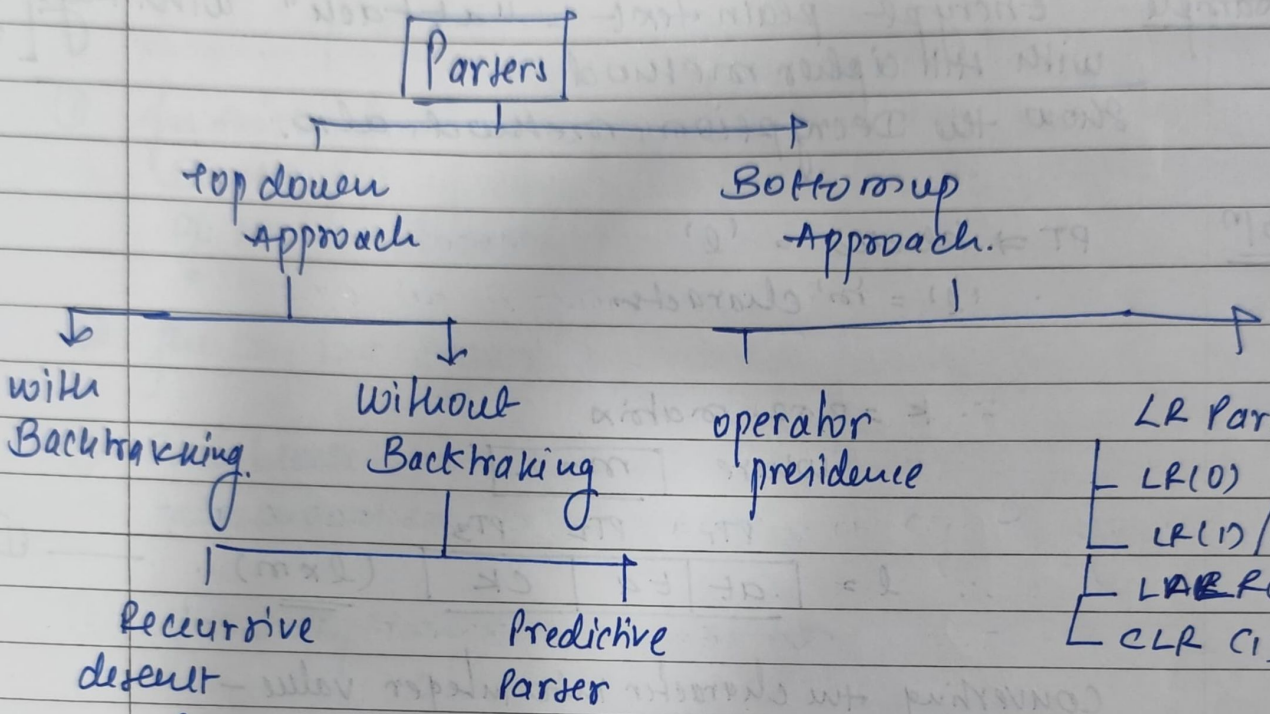
Follow(A) = {a, b}

Follow(B) = {b, a}

top down – left-most derivation.
bottom up – right-most in reverse order

Date 5 | Aug | 24.
Page





Example As Derives

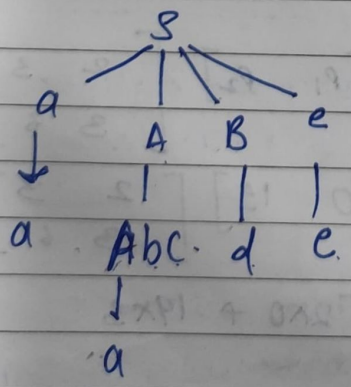
$S \rightarrow aABe$

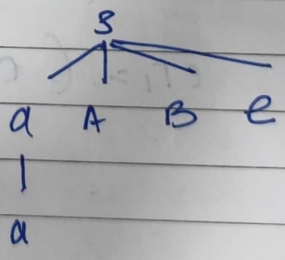
$A \rightarrow Abc \mid a$

$B \rightarrow d$

Input string → aabcde.

* It allows ambiguity.





① More than 2 operator then must

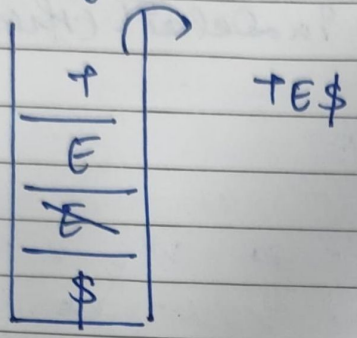
② More than 2 occurance of 2 operator then must see Attributive of that operator.

Sangam

CC

Date 8/Aug/24.
Page

* Stack for parse tree



First & follow set-

Rules for first-

① First - If α is the string of grammar symbol first of α, first(α) is the set of terminal that begin the string derived from α.
If $\alpha \rightarrow \epsilon$ or $\alpha \xrightarrow{*} \epsilon$ then ϵ is also in first(α).

② Follow - for non-terminal A follow(A) is the set of terminals that can appear immediately to right of A in some sentencial forms.

To compute first(x) for all grammar symbol x. Apply following rules until No more terminals or ϵ can be added to first set

Rule ① If x is a terminal then first(x) = {x}
(x: terminal)

Rule ② If $x \rightarrow \epsilon$ is a production then add ϵ to first(x)

Sangam

CC

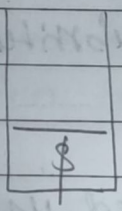
Date 12/Aug/24
Page

Parsing Tree Table

	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→id			F→(E)		

① First()
② Follow()

Stack



Lookahead symbol → (7) → that is LL1

Stack	Input	output
$	id+id*id$	
$E	id+id*id$	
$E'T	id+id*id$	E→TE'
$E'T'F	id+id*id$	T→FT'
$E'T'id	id+id*id$	F→id
$E'T'	+id*id$	
$E'	+id*id$	
$+TE'	+id*id$	
$TE'	id*id$	
$FT'E	id*id$	
$idT'E	id*id$	
$T'E	*id$	
$*FT'E	*id$	
$FT'E	id$	
$idT'E	id$	
$T'E	$	

Sangam

Rule (3) If α is a non terminal & $\alpha \to \gamma_1 \gamma_2 \gamma_3 \ldots \gamma_k$ then place A in first(α) if for some γ_I A is in first(γ_I).

Follow Rule — Never contain ϵ

(*) To compute follow of A for all NON-terminals (A) Apply following rules until Nothing can be added to follow sets.

Rule (1) - Place $ in ~~start~~ follow of start symbol.

Rule (2) - If there is a production of form $A \to \alpha B \beta$ then everything in first of β except ϵ is placed in follow of B.

Rule (3) If there is a production $A \to \alpha B$ or $A \to \alpha B \beta$ where first(β) contains ϵ then everything in follow of A is in follow of B.

Examples (1)

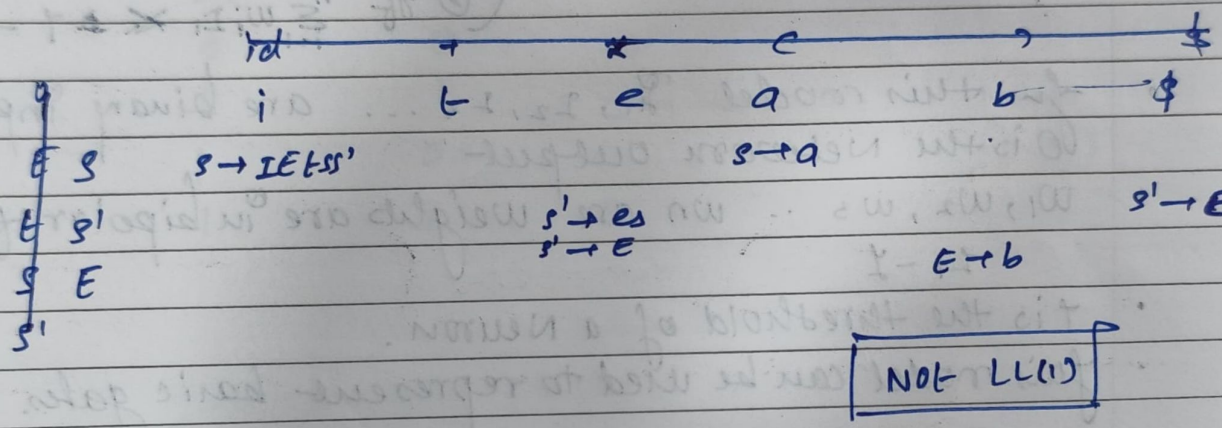
	first()	follow()
$S \to ABCDE$	{a, b, c}	{$}
$A \to a/\epsilon$	{a, ϵ}	{b, c}
$B \to b/\epsilon$	{b, ϵ}	{c}
$C \to c$	{c}	{d, E, $}
$D \to d/\epsilon$	{d, ϵ}	{e, $}
$E \to e/\epsilon$	{e, ϵ}	{$}

Sangam

Date
Page

Qw State the following grammar is LL1 or not.

	first()	follow()
$S \to i E t S S' / a$	$\{i, a\}$	$\{\$, e\}$
$S' \to eS / \in$	$\{e, \in\}$	$\{e, \$\}$
$E \to b$	$\{b\}$	$\{t\}$



NOT LL(1)

CC

Date
Page

* Recursive Descent parser

- It is build from a set of recursive procedures where each such procedures implements one of the non-terminals of grammar.
- Thus the structure of resulting program closely mirrors that of grammar it recognizes.

Ex

$E \rightarrow iE'$

$E' \rightarrow +iE' \mid E$

```
E()
{ if (l == 'i')
    { match ('i');
      E'();
    }
}
```

```
E'()
{ if (l == '+')
  {
    match ('+');
    match ('i');
      E'();
  }
  else
    return;
}
```

```
match (char t)
{
  if (l == t)
  l = getchar();
  else
  printf ("Error");
}
```

```
main()
{
  E();
  if (l == '$')
  printf ("Parsing success
           fail");
}
```

Sangam

CC

2 CFG are equal when they produces same strings

CFG — NO 2 adjacent non-terminals
└ NO ε-production

$A \longrightarrow \alpha \qquad \alpha \longrightarrow (VUT)^*$

ex $A \longrightarrow BC$ ✗ NO 2 adj Non terminal

* In operator precedence parsing we define 3 disjoint precedence relations between certain pairs of terminals.

$\lessdot \cdot \quad \doteq \quad \cdot \gtrdot$

This precedence rel have some meaning –

① $a \lessdot b$ a yeilds precedence to b.
b is having more priority than a.

② $a \doteq b$ same precedence

③ $a \gtrdot b$ a takes precedence over b.

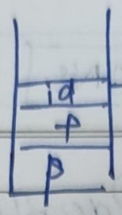
	id	+	*	$
id	—	$\cdot\gtrdot$	$\cdot\gtrdot$	$\cdot\gtrdot$
+	$\lessdot\cdot$	$\cdot\gtrdot$	$\lessdot\cdot$	$\cdot\gtrdot$
*	$\lessdot\cdot$	$\cdot\gtrdot$	$\cdot\gtrdot$	$\cdot\gtrdot$
$	$\lessdot\cdot$	$\lessdot\cdot$	$\lessdot\cdot$	Accept

g(n) →

id * id
id – id
id +

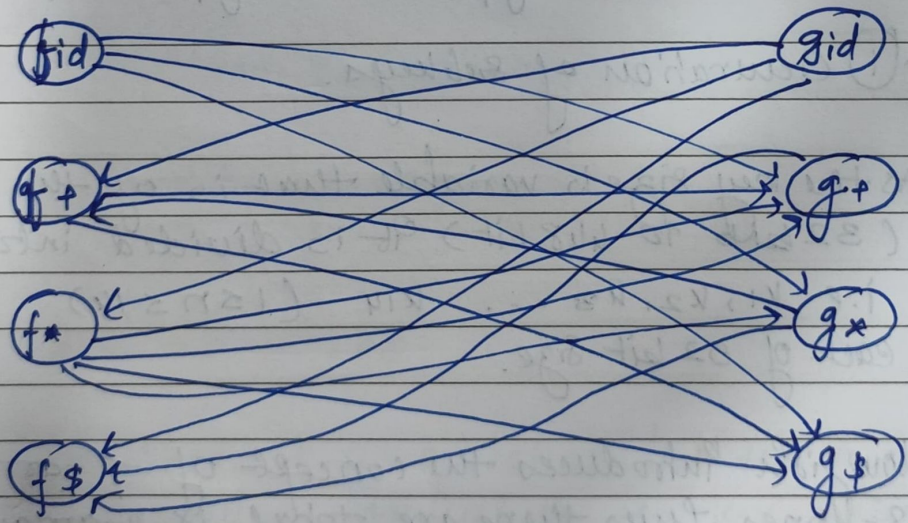
Rows → top of the stack
Column → input pointer.
f(n)

Blank entry considered as Error.



Algorithm

* Let 'a' be the topmost terminal symbol on the stack and Let "b" be the symbol pointed by input pointer.
* If $a < b$ then push 'b' on to the stack.
  $a = b$
* Increment the input pointer to the next input symbol.
* If $a > b$ the pop the stack until the top stack terminal is Related by $<$ to the terminal most Recently poped.



fid → g* → f+ → g+ → f$ length = 4

gid → f* → g* → f+ → g+ → f$

	id	+	*	$
f	4	2	4	0
g	5	1	3	0

CC

LR Parscer (LR)(k) – (shift reduce parser)
(Push) (POP)

- LR(0) Parser
- SLR(1) parser
- LALR(1) lookahead LR parser
- CLR(1) canonical LR

LR(0) items are used (LR(0), SLR(1))

LR(1) items are used. (LALR(1), CLR(1))

① Augmented grammar

S' → S   start / symbol
S → AB   when to reduce.?
A → a
B → b

② Canonical LR(0) items. → Defines how much of it we have seen.

S → •AB   If there is (•) opreater this means we have not seen AB both

S → A•B → we have seen A But not B.

S → AB• – final item LR(0).

* LR(1) Parcer

State No	Action		Goto	
	terminals		variables	
	a	b	A	B
0				
1	shift			
2	Reduce			
3				

LR
- → closure
- → goto

goto operator.
S → •AB
A → •a } shifting operation

represented by S_i → (state number)

CC

Date
Page

① Closure operation - In LR(0) parsing closure & goto operation are used to manage parsing operation.

Given a set of LR(0) items the closure operations adds all items that can be derived from the current items by expanding the non-terminal following (•) operator

② Goto - Given a set of LR(0) items & grammar symbol (terminals or non terminals) the goto operation produces a new set ~~of a terminal~~ by moving (•) operator from one position to the Right for all given items where the dot is immediately before the given symbols.

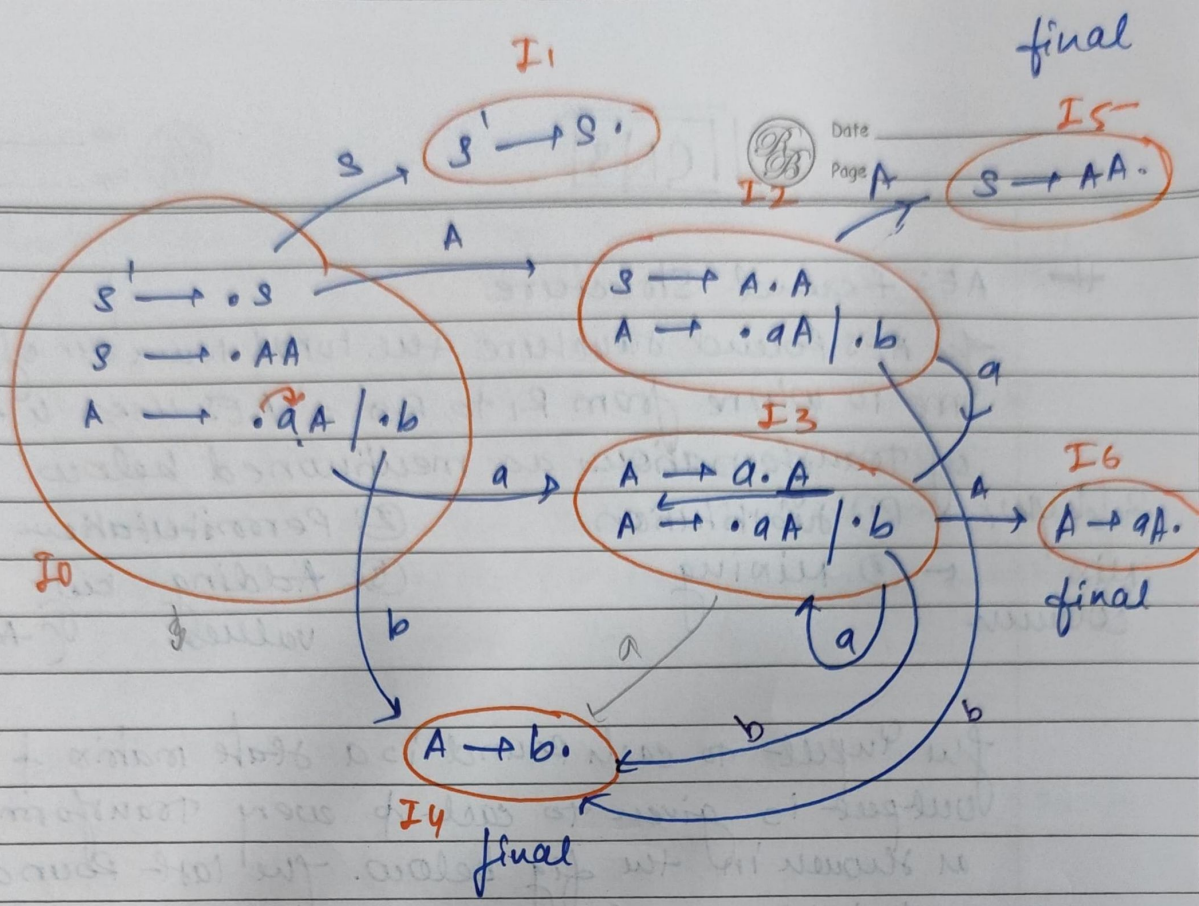
Question ① construct LR(0) parsing table for below grammar

$S \rightarrow AA$
$A \rightarrow aA \mid b$

→ $S \rightarrow AA$ ①
$S \rightarrow aA$ ②
$S \rightarrow b$ ③

} step ①.

Sangam



state	Action			Goto	
States	a	b	$	S	A
0	s_3	s_4		1	2
1			Accept		
2	s_3	s_4			5
3	s_3	s_4			6
4	r_3	r_3	r_3		
5	r_1	r_1	r_1		
6	r_2	r_2	r_2		

Reduce

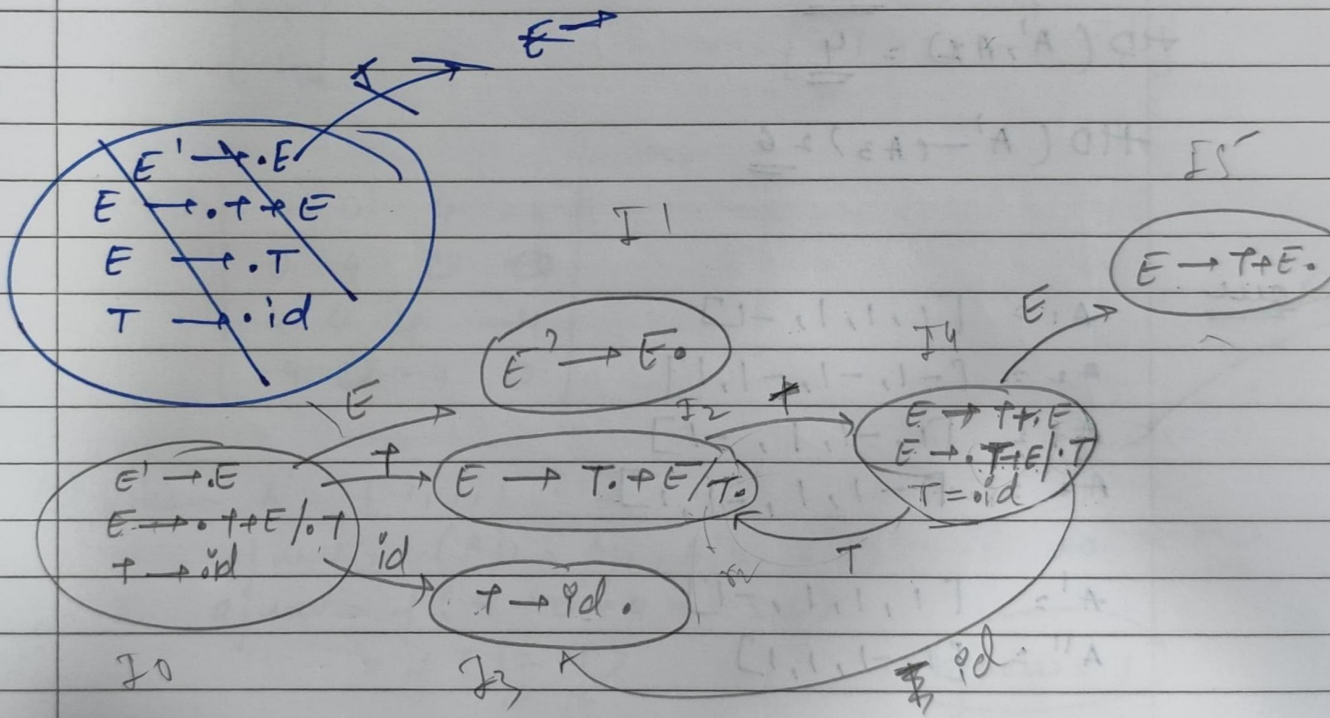
CC

Date 8-09-21 (2)
Page

Question State whether the following grammar is LR(0)

$E \rightarrow T + E / T$
$T \rightarrow id$

→ $E \rightarrow T + E$
$E \rightarrow T$
$T \rightarrow id$



State	Action			Goto	
	+	id	$	T	E
0		S_3		2	1
1			Accept		
2	S_4	r_2	r_2		
3		r_3			
4					
5					

Sangam

The given grammar is Not LR(0) as it is S-R conflict.

CC

Date
Page

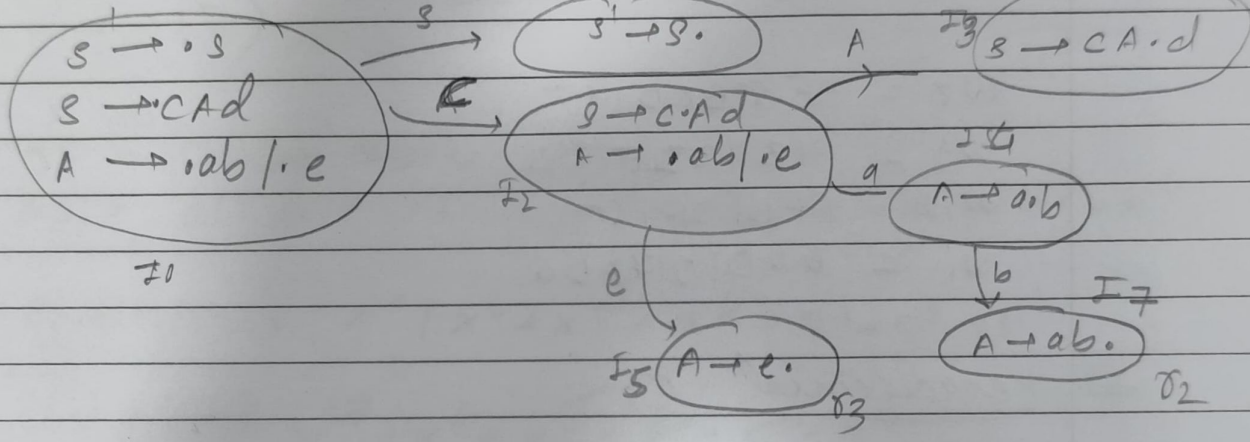
Q11 Consider the grammar given below (LL(1), LL(0), SLR(1))

$S \rightarrow CAd$

$A \rightarrow ab \mid e$

String : ced

Parse the input string using SLR(1) parsing.



State	Action						Goto	
	a	b	c	d	e	$	S	A
0			S_2				1	
1						Accept		
2	S_4				S_5			3
3				S_6				
4			S_7					
5	r_3	r_3	r_3	r_3	r_3	r_3		
6	r_1	r_1	r_1	r_1	r_1	r_1		
7	r_2	r_2	r_2	r_2	r_2	r_2		

SLR(1).

→ LL(0) ✓

→ given grammar is SLR(1) as there is no conflict

→ LL(1) ✓ also

↳ First, Follow

Sangam

CC

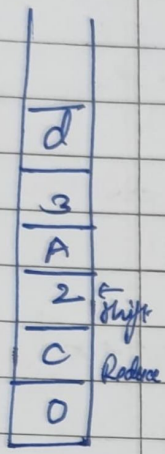
A → a b | e
Date Page
1 2 3

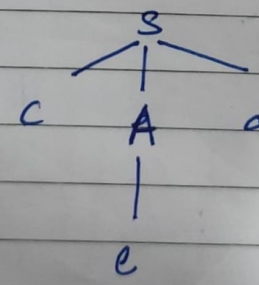
Last solved example. (Parsing) Part II

Input string : ced

Stack	Input	Action / Output
0	ced$	shift 0
0C2	ed$	shift C → S2
0C2e5	d$	shift e → S5
0C2A3	d$	Reduce A → e.
0C2A3d6	$	d → S6
0S1	$	Reduce. S → CAd.

String is Accepted.
Because S on 1 only needed to accept string by $.





Sangam

LR(0) & SLR(1)
LR(0)
$S \to .AA$

CLR & LALR
LR(1) item
LR(0) item with a lookahead symbol.

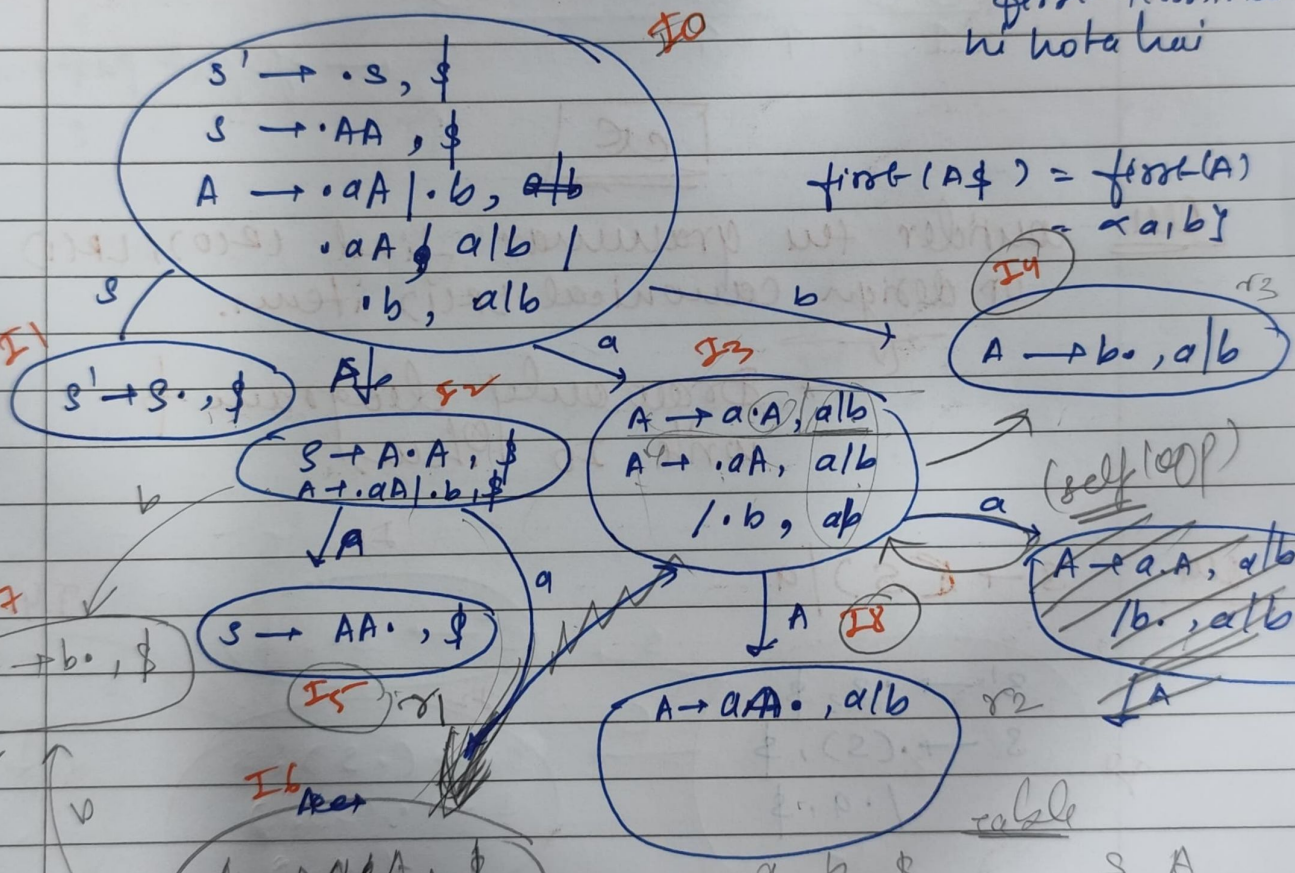
Canonical LR & Lookahead LR

In CLR & LAR parsing methods canonical (R11) are used or utilised to design parsing table.

Problem. construct CLR(1) / parsing table → Also design LALR grammar

$S \to AA$
$A \to aA \mid b$

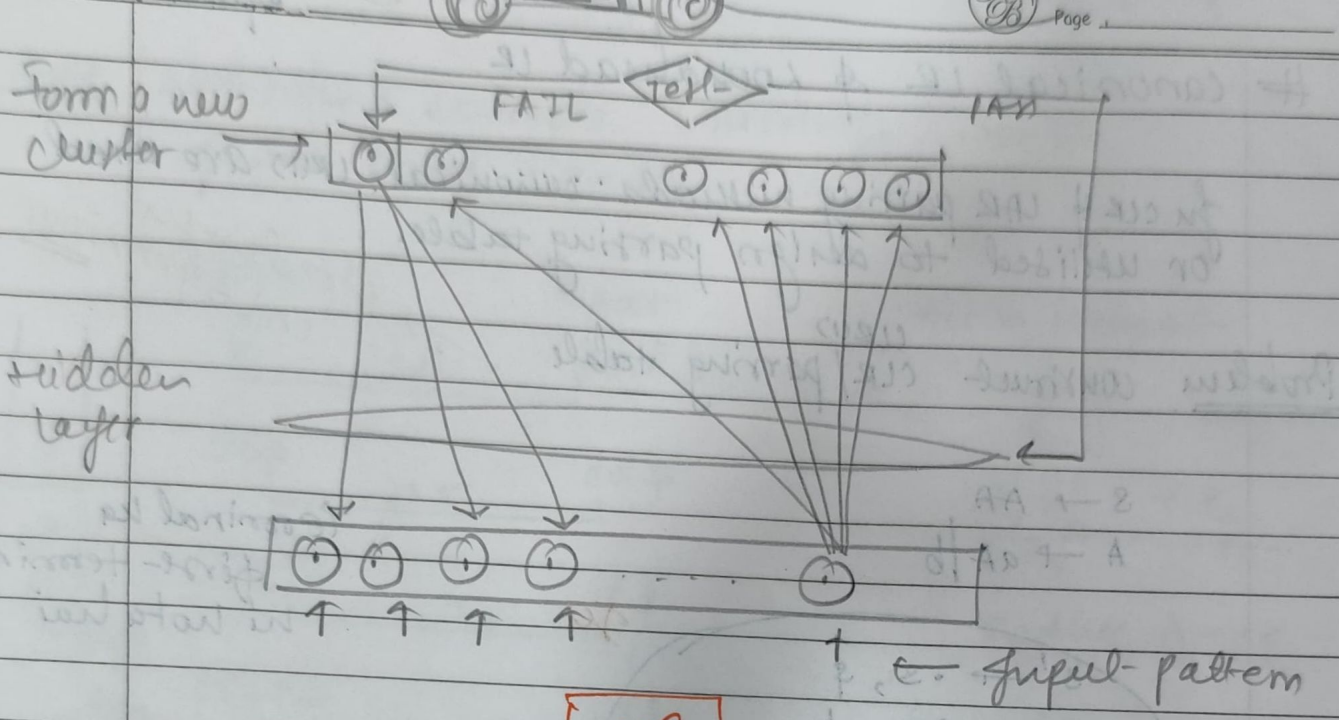
terminal ka first terminal hi hota hai



CLR parsing table

	a	b	$	S	A
0	s_3	s_4		1	2
1			Accept		
2	s_6	s_7			5
3	s_3	s_4			8
4	r_3	r_3			
5			r_1		
6	s_6	s_7			9
7			r_2		
8	r_2	r_2			
9			r_2		

Sangam



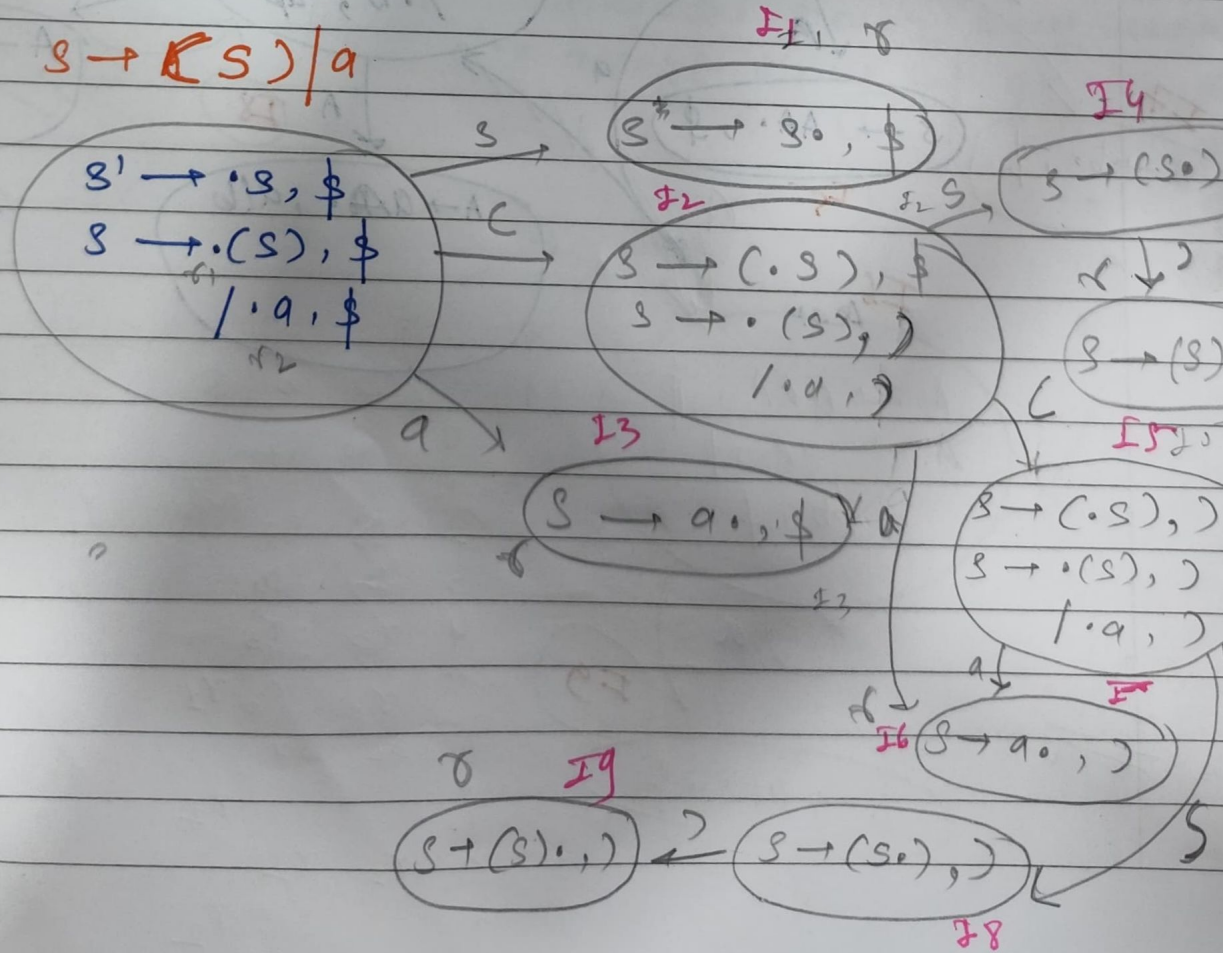
CC

Q.1 consider the grammar find LR(0), LR(1) or design canonical LR(1) items.

→ Draw only diagram & write I0 items.

Q.1 $S \to (S) \mid a$

→



Sangam

terminal | Nonterminal

	(	)	a	$	S
0	S2		S3		1
1				Accept	
2	S5		S6		4
3				r2	
4		S7			
5	S5		S6		8
6		r2			
7				r1	
8		S9			
9		r1			

NOW for LALR(1)

I3	I6
I7	I9.
I4	I8

} can be combined. all lookahead should be different

	(	)	a	$	S
0	S2		S36		1
1				Accept	
2	S5		S63		48
36		r2		r2	
48		S79			
79		r1		r1	
5	S5		S6		48

therefore LALR(1) is accepted.