WWW.GRADESETTER.COM

Compiler

(i) LL(1)

(a) $A \to \alpha_1 / \alpha_2$ then, if $first(\alpha_1) \cap first(\alpha_2) \neq \phi \Rightarrow$ Not LL(1)

(b) $A \to \alpha_1 / \epsilon$ then if $First(\alpha) \cap Follow(A) \neq \phi \Rightarrow$ Not LL(1)

LL(1) is always LR(1)

(2) LR(K) — "If LR(0) then it is also SLR(1)"

(a) SLR(1) :-

SR conflict

$A \to \alpha \cdot a \beta$
$B \to \gamma \cdot$

if $\{a\} \cap follow(B) \neq \phi$
= S/R conflict
= Not SLR(1)

RR conflict

$A \to \alpha \cdot$
$B \to \gamma \cdot$

if $Follow(A) \cap Follow(B) \neq \phi$
= R/R conflict
= Not SLR(1)

Every LR(0) is SLR(1)

(b) LR(0) :- same rule

" Not LR(0)

	a	b	c	$
i	S/Ri	Ri	Ri	Ri

same rule

" Not LR(0)

	a	b	c	$
i	R1/R2	R1/R2	R1/R2	R1/R2

(c) CLR(1) or LR(1) :-

$A \to \alpha \cdot a \beta, b$
$B \to \gamma \cdot, a$

⇒ SR conflict
⇒ Not CLR(1)

$A \to \alpha \cdot, a$
$B \to \gamma \cdot, a$

⇒ RR conflict
⇒ Not CLR(1)

Every LALR(1) is CLR(1) — same rule

(d) LALR(1) :-

" — Not SR conflict

" — same rule

→ Not RR conflict ⋮ still you have to check → Yes/No RR conflict

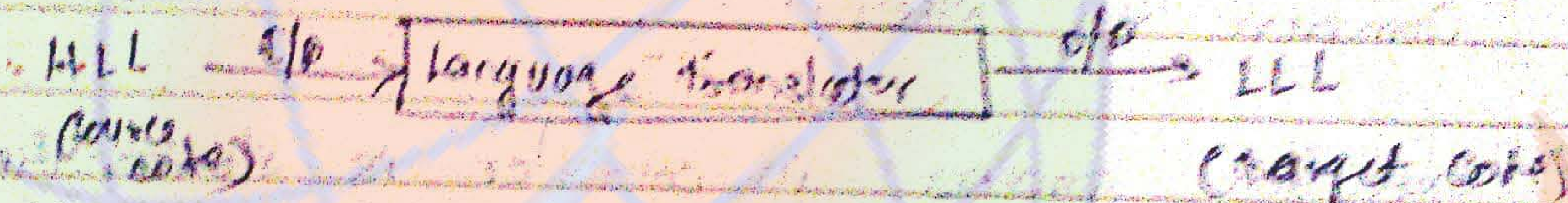
WWW.GRADESETTER.COM

Compiler Design

{ CO = 4-6 }
{ T.O.C = 8-11 }

- Phase of a compiler
- Lexical Analysis
- Syntax Analysis
- Semantic Analysis
- Intermediate Code Generation
- Code Optimization
- Code Generation

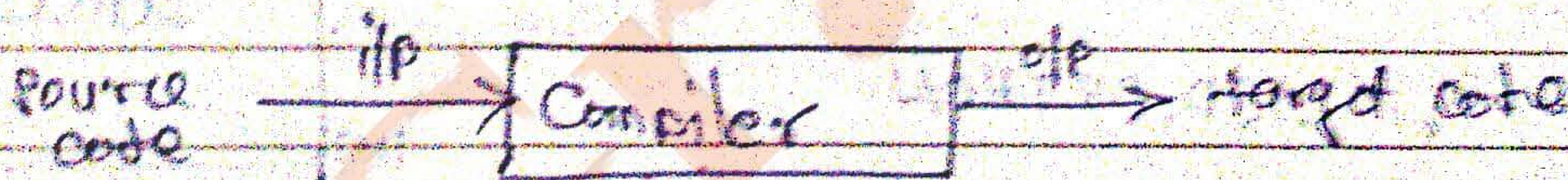
1) Language translator:-

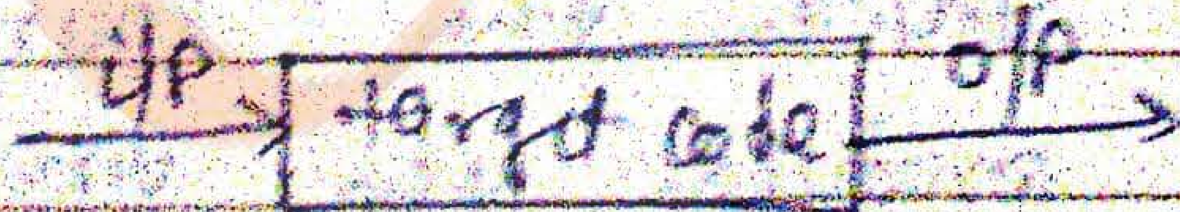


Language translator takes source code as input and produces target code as output.

2) The types of language translator:-
   a) Compiler
   b) Interpreter
   c) Assembler

3) a) Compiler





execute the (whole) all at a time

eg C, C++

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Compiler takes the source code as input and produces the target code as output. then

This target code will be called by the user to process the inputs for producing the output.

A compiler execute the entire code at a time. If any error occurs at any line, all of them will be given at a time.

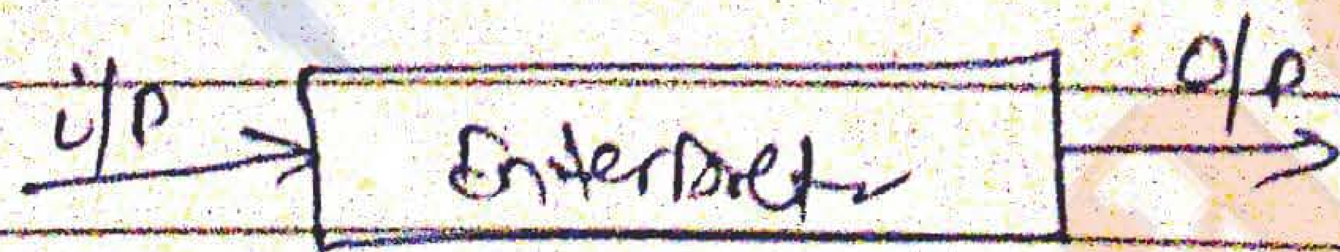
- Error diagnosis is not that much easy as compare to other's.
- Execution process in compiler is slower than other's

But

- Output generation by compiler is faster.

eg. C, C++

b) Interpreter:-



eg. LISP, PYTHON

Interpreter takes the source code as input and produces the direct output. It will not produce any intermediate language

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

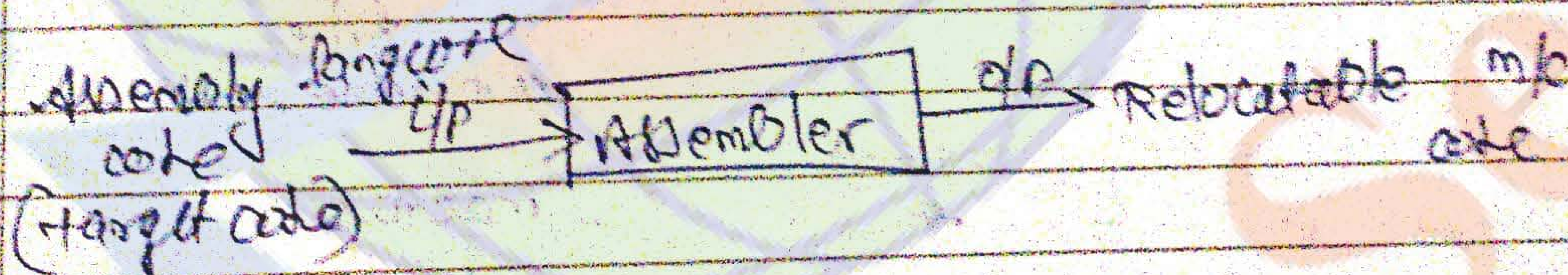
- Interpreter: execute the code line by line
  if any error occurs at any line, immediately that error will be produce.
  Untill we resolve that error the interpreter will not move to the next line.

- Error diagnosis in interpreter is easy.

- Execution process in interpreter is faster. That output generation is slower then the Compiler

  eg. LISP, PYTHON

c) Assembler:-



It takes the assembly language code as input and produces relocatable m/c code as output which has to be loaded into main memory ready for execution

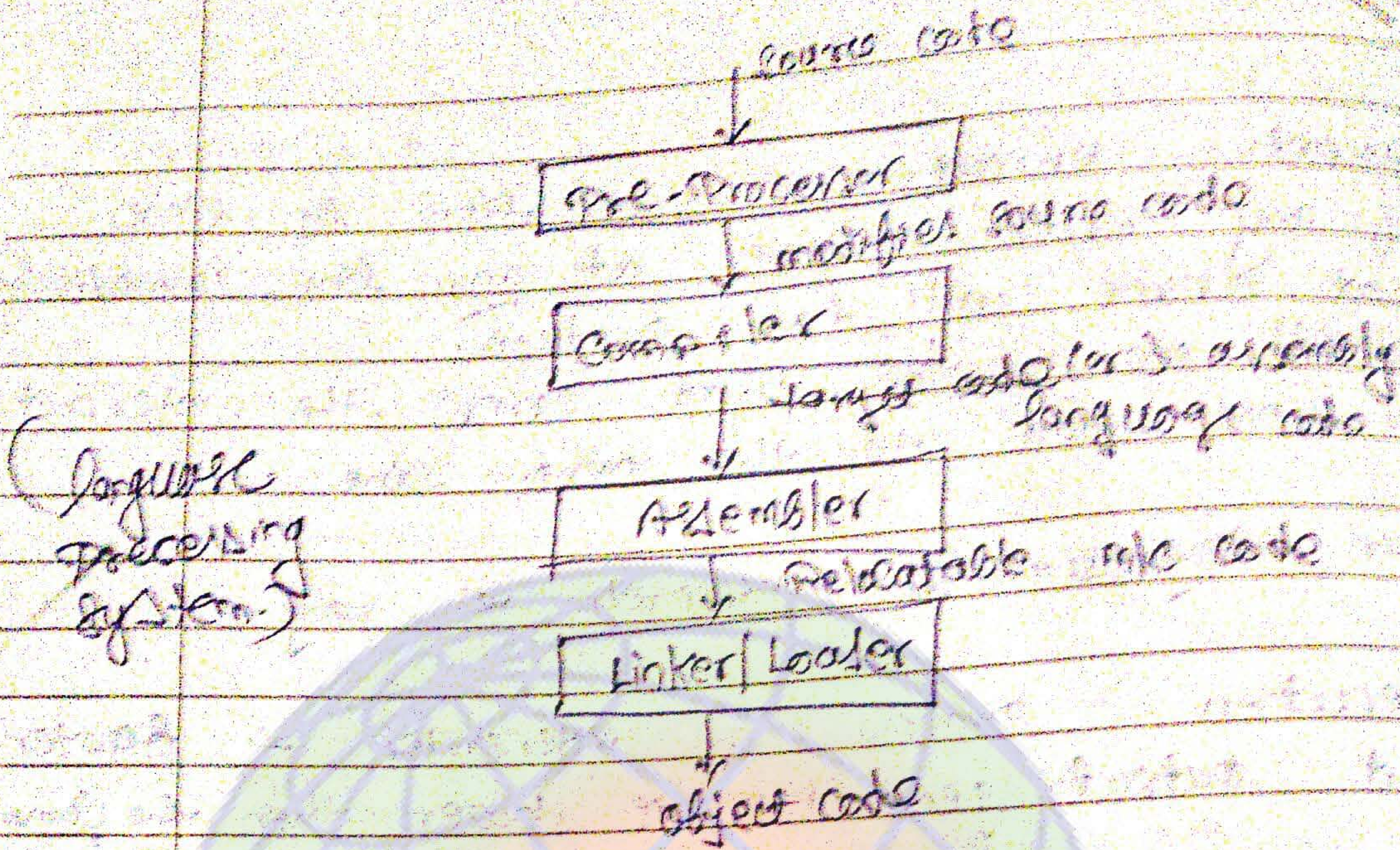
4) Language processing System:-

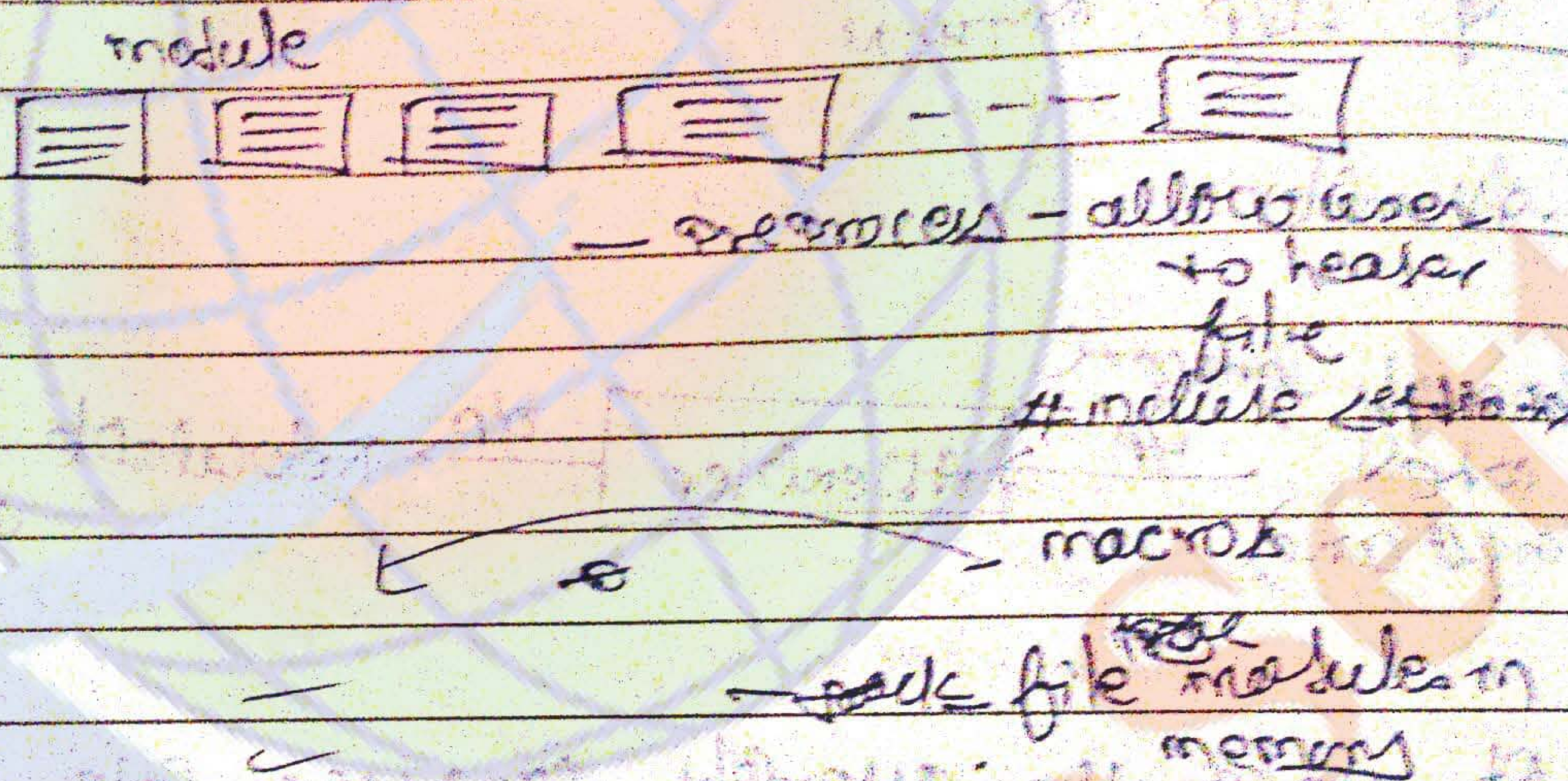
Pre-Processor - takes the source code as input and produces modifies source code as output

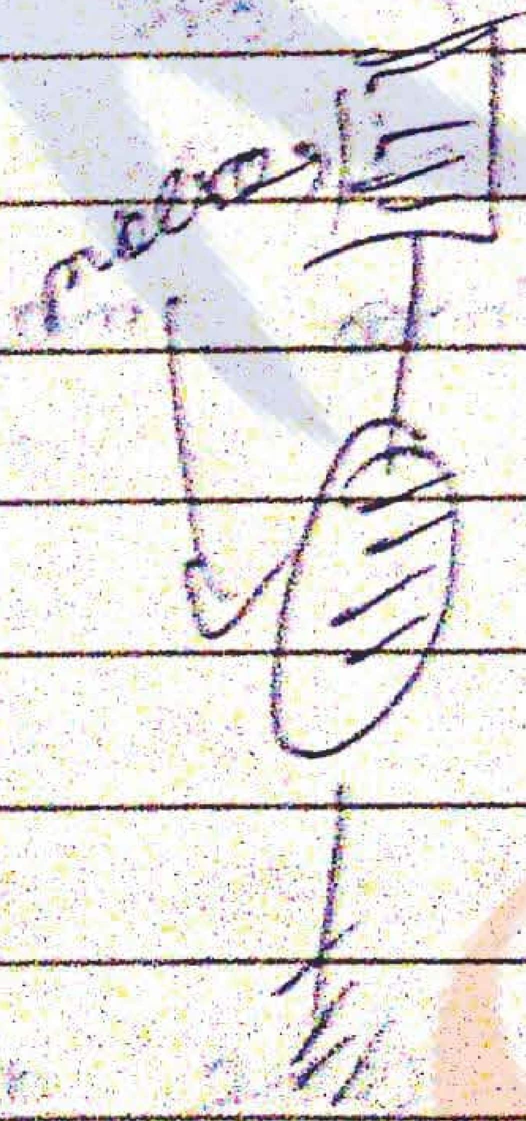
Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM







• Preprocess — Preprocess takes the source code as input and produces modified source code as input

— If the program is too large it may

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

be divides into small pieces and they will be stores in different file.

- The pre-proces will takes care of all these modules.

- Pre-Processor allow the user to use header files and macros.

- A macro is a set of instructions which are repeatedly used in the program.

- Pre-Processing is an optional
  eg. the languages which do not support #include < > they do not requires Pre-Processing.

Compiler - compiler takes the modifies source code as input and produces the target code as output.

- Compilation is also an optional
  eg. script languages like java script.
  HTML
  Assembly language like 8085 8086
  they not requires compile

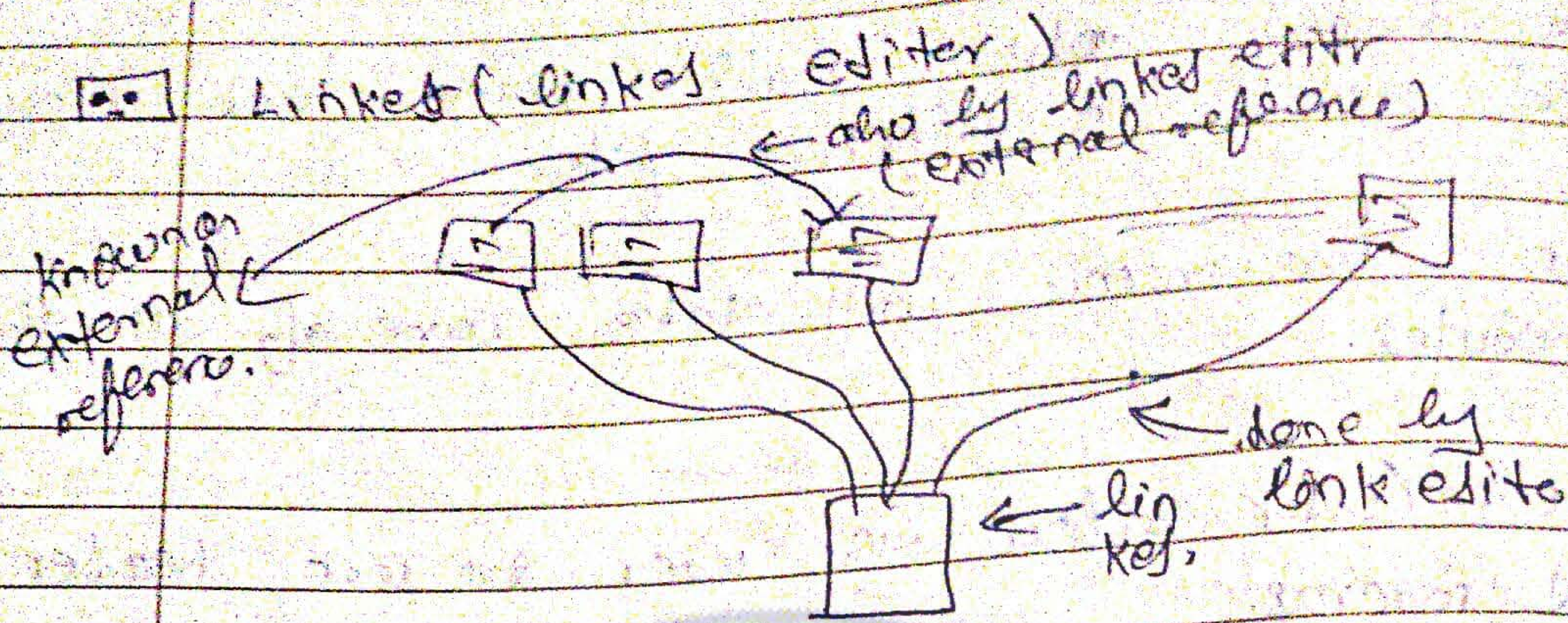
Assemble -

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Linker (linker editer)



- linker editr - If the program is too large, it may be divided into small piece and will be stored in different files.
- The linker links all these module as a single mode.
- The link editer resolves external reference.
- If a code in one files reffers allocation in another files is called as external refrence.

Loader:-

To execute the program, it must be loaded in main memory. This will be done by loader.

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

5) Phases of Compiler:

source code
↓
[Lexical Analysis or Scanner] → scans every character
↓ tokens
[Syntax Analysis]
↓ Parse tree
[Semantic Analysis]
↓ Annotated Parse tree
[Intermediate code generation]
↓ Intermediate code
[Code optimization]
↓ optimized code
[Code generation]
↓
target code

a) Lexical Analysis.

a = b + c * 10

- a → identifiers
- = → operator
- b → id
- + → op
- c → id
- * → op
- 10 → constant

Lexical Analysis takes the source code as input and divides that code into tokens.

a = b + c * 10 or a = | b | + | c | * | 10 |

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

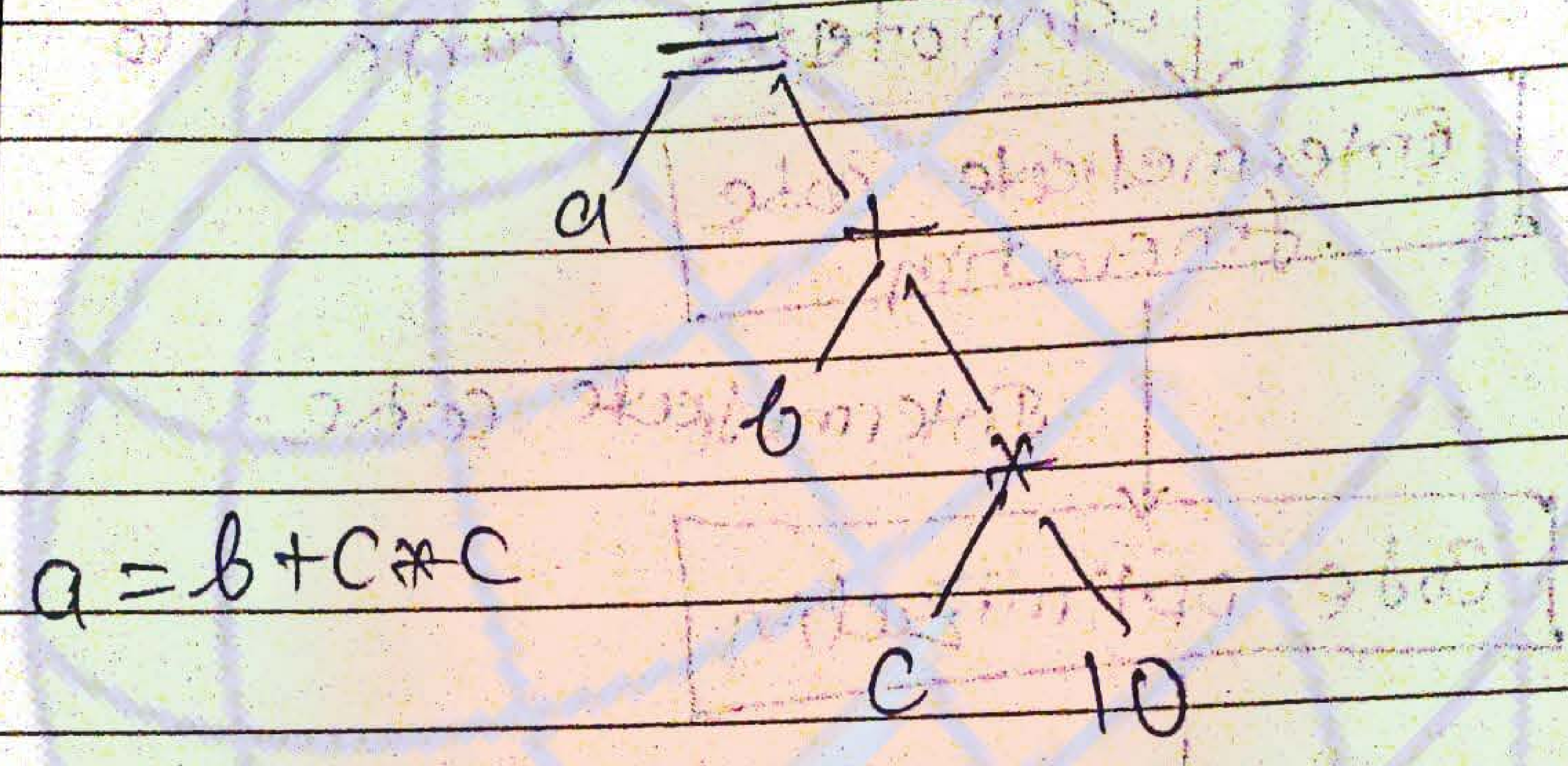
Page No.
Date

b) Syntax Analysis:-

[CFG rule production is applied for]

The tokens generated by lexical analyser will be groups together in syntax analyser to make a parse tree.

$a = b + c * c$

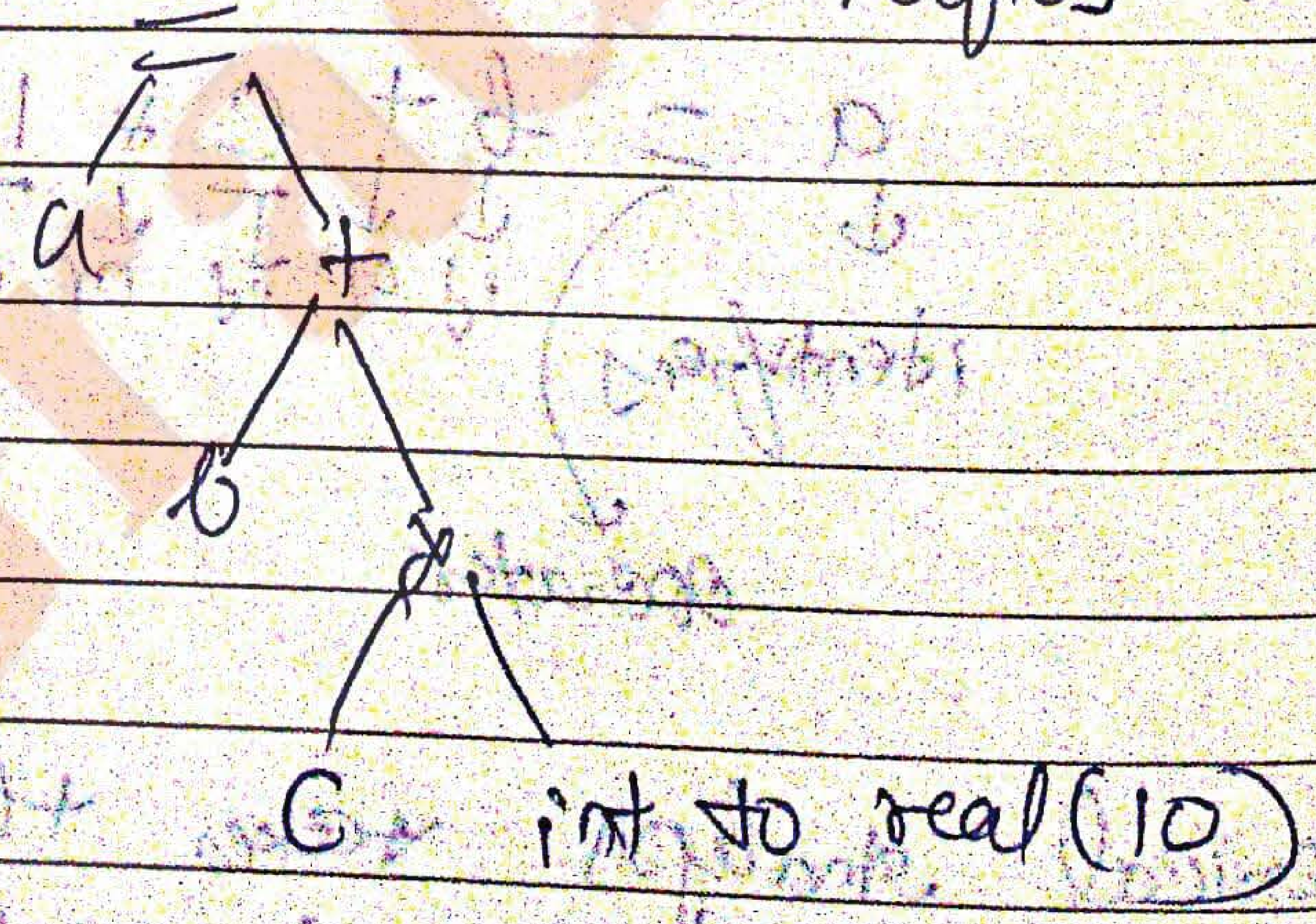


- The meaning of the source code will be verified in semantic analysis

~~c) Semantic Analysis~~

c) Semantic Analysis

~~meaning will be verifies~~ - Meaning will be verifies.



[Performs type Checking, i.e. type compatible or not]

c) Semantic
d) Inter

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

c) Semantic analysis
d) Intermediate code generation -

Various types of Intermediate code generat

- 3-address code → a = b op c

- **3-address code:**

$$a = b + c * 10.00$$

t_1 = int to real (10)
t_2 = c
t_3 = $t_2 * t_1$
t_4 = b
t_5 = $t_4 + t_3$
a = t_5

} inefficient code

Now,

t_1 = int to real (10)
t_2 = c * t_1
t_3 = b + t_2
a = t_3

} efficient code

- intermediate code makes the code generation process simple and easy

Scanned with CamScanner
WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

c) Semantic analysis

d) Intermediate code generation –

- Various types of Intermediate code generation

— 3-address code ⇒ a = b op c

- 3-address code:

$a = b + c * 10.00$

t_1 = int to real (10)
t_2 = c
t_3 = $t_2 * t_1$
t_4 = b
t_5 = $t_4 + t_3$
a = t_5

} inefficient code

Now:

t_1 = int to real (10)
t_2 = $c * t_1$
t_3 = $b + t_2$
a = t_3

} efficient code

– Intermediate code makes the code generation process simple and easy

Scanned with CamScanner
WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

- mostly we use 3-address code representation in intermediate code.

c) Code optimization:-
It is process of reducing the no. of instructions, without affecting the outcome of the source code.

t_1 = int to real(10)
t_2 = c * t_1
a = b + t_2

d) Code generation:-

```
MOV  R, #10.00
MUL  R, C
ADD  R, b
MOV  a, R
```

GradeSetter

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Lexical Analysis

The lexical analyser scans every character of a source and the following will be done.

- Divides into tokens
- Ignore blank space/ white space
- Ignore Comments.
- Creates a symbol table
- Produces lexical errors

a) Divides into tokens:-

~~Tokens~~ The terminology used in the lexical analysor are token

Tokens:- ~~The set of rule which describes the token~~

tokens can be KW, OP, punctuation marks,

tokens:- which describe the category of the input string

eg - keywords (KW), operator (OP), punctuation mark (PM), ~~const~~ constant, identifier (id)

An identifier can be any combination of alphanumeric symbols.
the first symbol of the identifier must be an alphabet or underscore.

- only special symbol allowed in identifier is underscore (_).

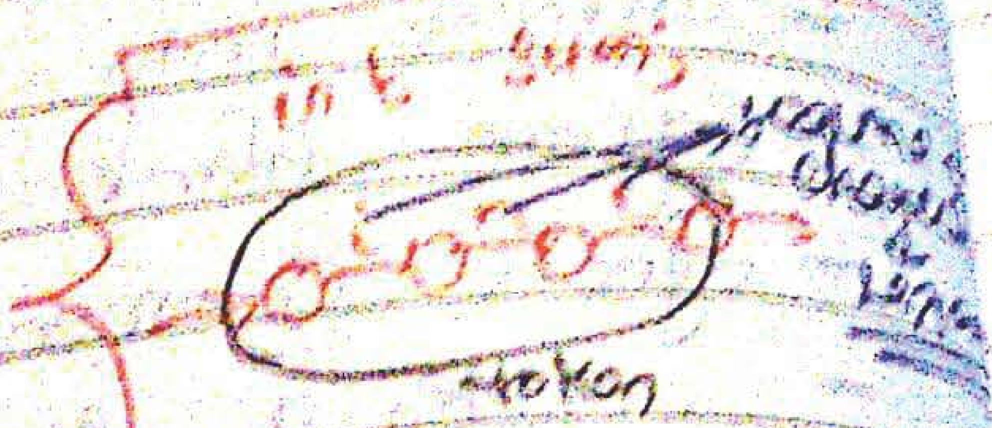
Patterns: The set of rules which describes the token.

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Lexeme: The sequence of the characters in the source program that matches with patterns of the tokens



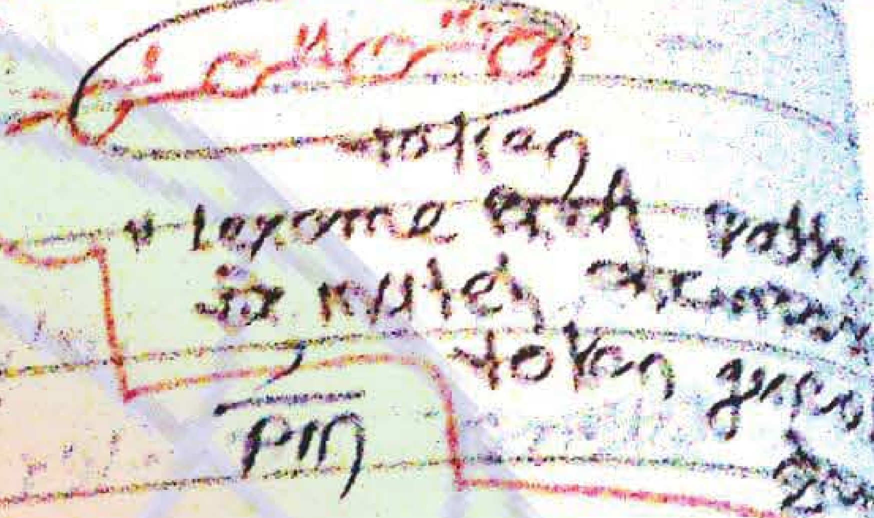
b) Ignore blank space.

int a , b ;
KW id PM id PM

- keywords
- identifiers
- punctuation marks

int a ; b
KW id PM id PM

Ignore space



c) Ignore comments

/* ——— */ → will not be converted into tokens

d) Create a symbol table
↓
is a data structure, which stores information

- Symbol table: A symbol table is a data-structure which contain information about variables and their attribute.
  eg. → name of the variable, datatype, value, size, location

e) Produces lexical errors.
Lexical errors: → Unterminated Comments /* ---
→ Nested comments
↳ /* —— /* —— */ —— */ Not allowed

Lexeme - sequence

Note: • The ... is sco...
• The ... finite
• Hand ...

Tokens

a) ~~key~~ check errors

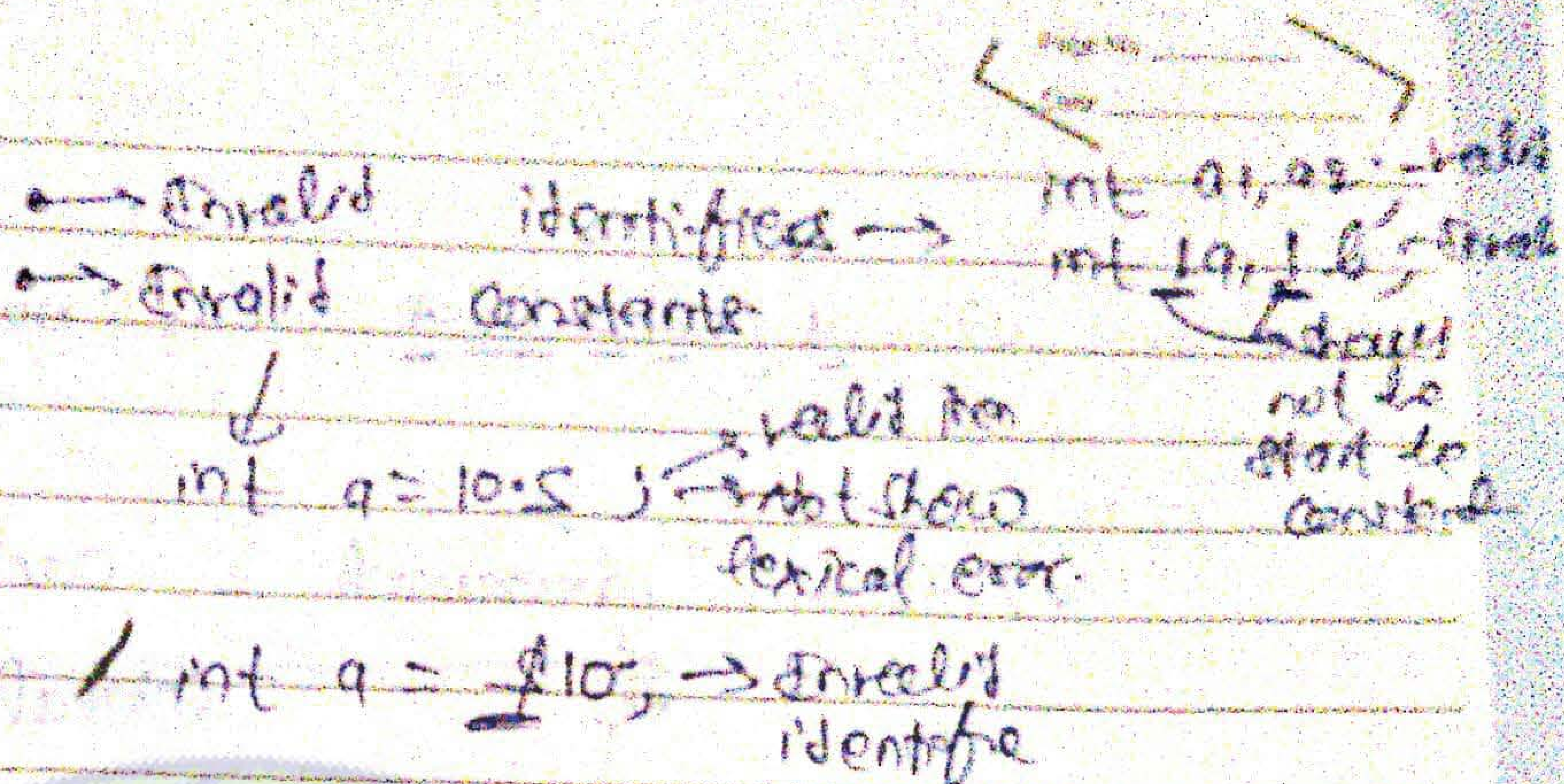
a)

b)

WWW.GRADESETTER.COM

Scanned with CamScanner

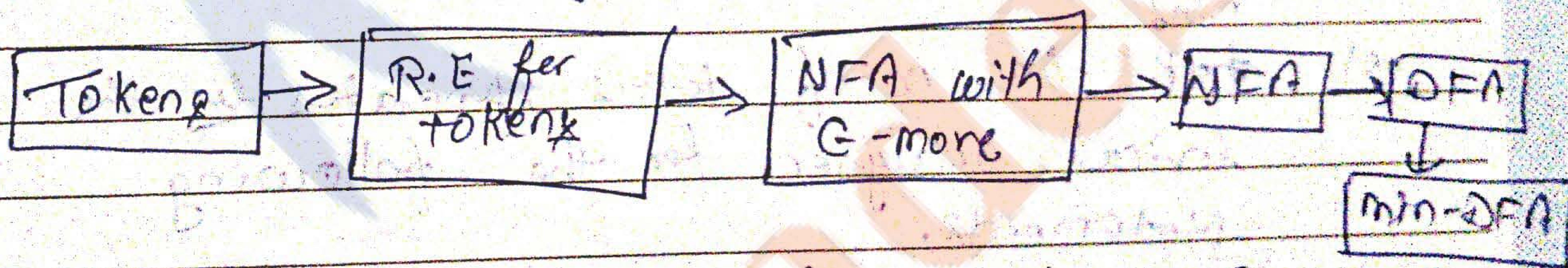
WWW.GRADESETTER.COM



Lexeme - sequence of characte

int a1, a2'; → lexeme
int 1a, 1b → Not a lexeme
int a = $10 → Not a lexeme.

Note:
- The other name of the lexical analyser is scanner
- The program used in lexical analysis is finite automata.
- Hand code design



Q) check weather lexical analyser can produce errors for the follawing statements.

a) iit a, b; (id, id, pm, id, pm) → No lexical errors.

b) fi (a > b) (id)
a = a+1
else
b = b+1 → No lexical error

Scanned with CamScanner
WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

c) a = b + cd # ; → No lexical error

d) Printf (" Amravati is capital of Andra);

← literal →

↳ Produces lexical error

because → दोनों side होना चाहीये brace.

e) int 1a, 1b ;

↳ Lexical error (invalid identifier)

f) int a = @10 ;

↳ Invalid constant (lexical error)

Q) Find the no. of tokens generates by the lexical analyser for the following statements.

a) a = b + c * d ;

id op id op id op id PM   Ans = 8

b) if (a > b)

a = a + 1

else

b = b + 1   Ans = 17

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

R1

c) for (i=0 , i<=n, i++) — No lexical error

d) for (i=0 ; i<=n ; i++) → 13
(for ↑ kw; i++) → R1)

e) int aaa; → 3

int aa a; → 4 (Note: space divides into tokens)

int a a a; → 5

f) Printf (" Visakhapatnam is a beautiful city ") ;
(Printf = id; the string = literals)

Ans - 5

Note: Printf & scanf is actually function, here we use it as an identifiers.

g) scanf ("%d %d" , a , b) ; (literal) Ans = 9

h) scanf ("%d" , &a) ; Ans = 8 (& = operator)

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Q) Consider the following C-Program and identify the compiler module about object lines.

a)
```
int main()
{
int i, n;
fro( i=0; i<=n; i++)
{

}
}
```

× a) lexical error only
✓ b) syntactical error only
× c) both lexical & syntactical error
× d) No compiler error

Hint: If you dont know exactly, go for eliminat

Q) An Ideal compiler should be
a) small in size
b) takes less than compilation
c) ~~produce~~ must produce an object code which executes faster
✓ d) all

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Ti

Page No
Date

a) In a compiler the keywords of a language will be recognized during
  - ✓a) lexical analysis
  - b) syntax analysis
  - c) code optimation
  - d) code generation

a) In a compiler the (module) grouping of characters into tokens will be done during
  - ✓a) scanner
  - b) parser
  - c) code optimizer
  - d) all

Hint: Ram

T1 a) An instruction in programming language i.e. the sequence of instructions prior to compiling is known as
  - a) Literal
  - b) Procedure
  - ✓c) macro
  - d) label

Hint: macro - it will copy the method

a) The datastructure in a compiler, used for storing ~~about~~ the information about the variables and their attributes is symbol table

a) A link editor is a program that
  - ✓a) matches external names of one program with locations in other program

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

~~b)~~ matches the parameters of macro definition with location

Q.) A system program, that sets up an executable program in main memory ready for execution is
a) Assembler
~~b)~~ loader
c) compiler
d) linker

Q.) In lexical analysis, the moder computer computer language such as JAVA, needs the power of which one of the following machine model in necessary and sufficient sense -
(a) finite automata
b) push down automata
c) T.M
d) All

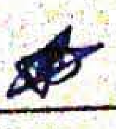
Note: If not mention sufficient, the go for (d) all

Syntax

The ...
be grou...
a par...
To ...
some ...
will he...

Context

If ...


Ana...

A ...

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

b) matches the parameters of macro definition with location

Q) A system program, that sets up an executable program in main memory ready for execution is:

a) Assembler
b) loader
c) compiler
d) linker

Q) In lexical analysis, the modern computer computer language such as JAVA, needs the power of which one of the following machine model in necessary and sufficient sense -

(a) finite automata
b) push down automata
c) T.m
d) All

Note: If not mention sufficient, then go for (d) all

★ Syntax

The ... be grou... a ga... To ... some... will ...

★ Context ...

If ...

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

★ Syntax Analysis:

The tokens generated by lexical analyser will be grouped together in syntax analyser to form a parse tree.
To form such a tree there must be some rules are required, and those rules will be supplied by context free grammar (CFL)

$A \rightarrow \alpha$

where $A \in V$, (only non terminal)

$\alpha \in (V \cup T)^*$ → both

★ Context free grammar (CFL)

If every production of the grammar is of the form

$A \rightarrow \alpha$

where $A \in V$, $\alpha \in (V \cup T)^*$

★ Ambiguity:-

— more than one parse tree possible

A grammar is said to be ambiguous, if it can derive at least one string, which is ambiguous.

$L(G_1) = \{ \omega_1, \omega_2, \ldots \}$

If one string is ambiguous, then the grammar is ambiguous.

Scanned with CamScanner
WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

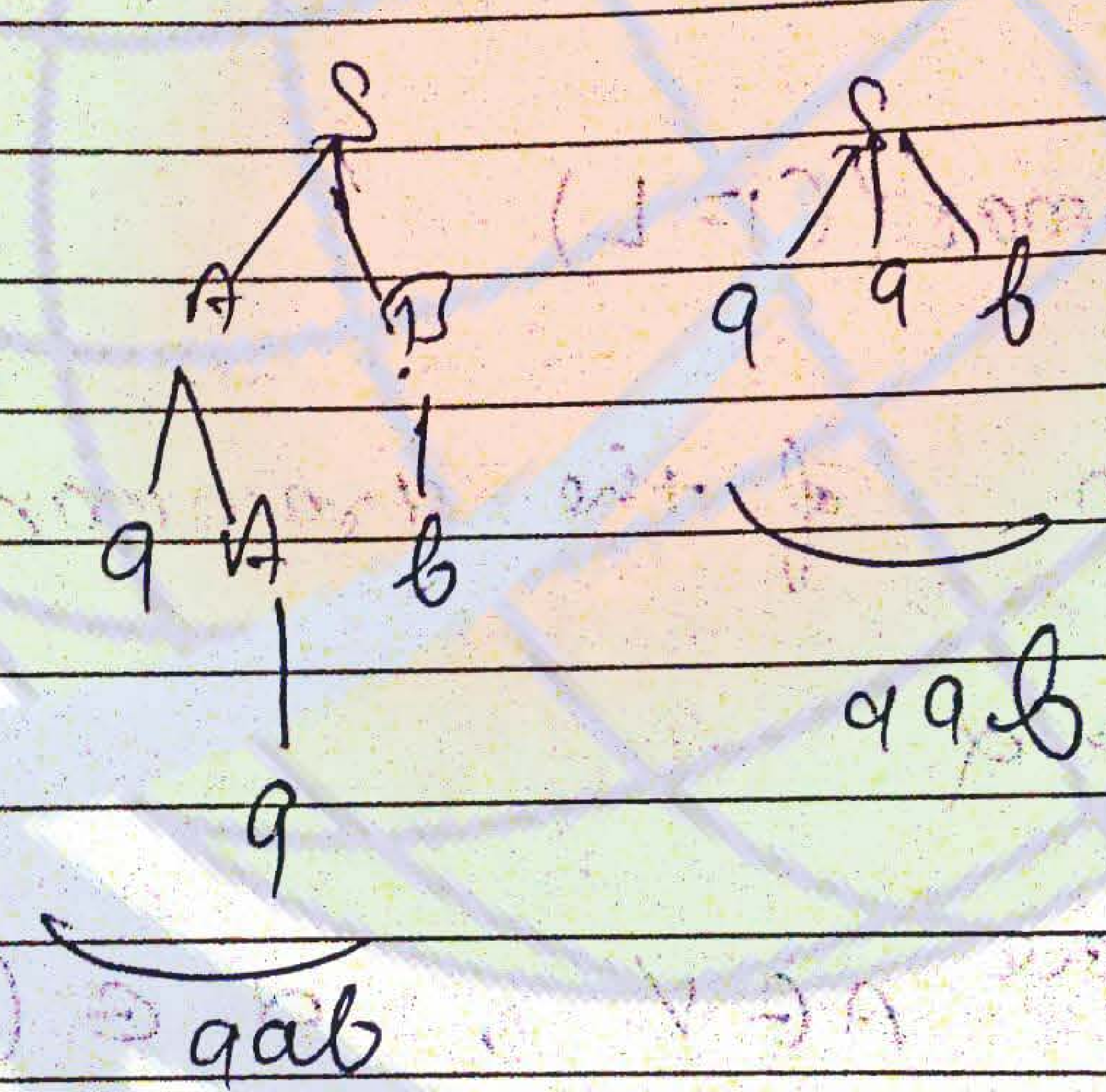
• A string $w \in T^*$ is said to be ambigious, if it has more then one different parse tree.

Q) Is this grammar ambigious?

$G = \{ S \to AB / aab, \; A \to aA / a, \; B \to bB / b \}$

Soln

$S \to \underline{A} \; B$



So, Ambigious

G is ambig

Q) v.vi

Note

• An ambigious grammar is not suitable for any kind of parsing, except backtracking, shift reduce and operator precedence parsing.

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Page No.
Date

Elimination of Ambiguity :-

Therefore, we must eliminate the ambiguity from a grammar.

But there is no procedure to eliminate the ambiguity from a grammar.

That's why, elimination of Ambiguity from a grammar is undecidable.

eg1) $G = \{S \rightarrow AB \mid aab, A \rightarrow aA/a, B \rightarrow bB/b\}$

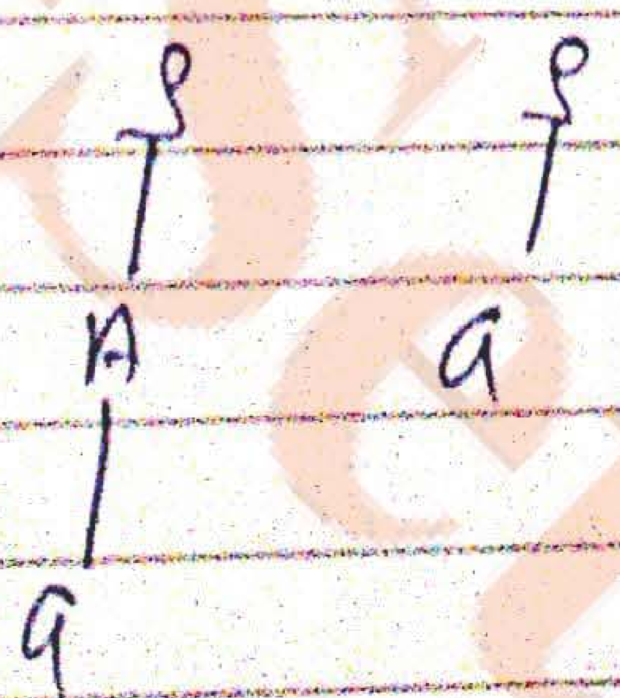
i.e. ambiguous grammar

to eliminate.

$G' = \{S \rightarrow AB, A \rightarrow aA/a, B \rightarrow bB/b\}$

i.e. unambiguous grammar

eg2) $G = \{S \rightarrow A/a, A \rightarrow a\}$

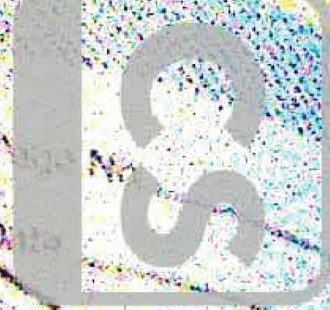


So, this is ambiguous

So,

$G' = \{S \rightarrow a\}$

→ This is unambiguous


Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Elimination of Ambiguity :-

Therefore, we must eliminate the ambiguity from a grammar.

But there is no procedure to eliminate the ambiguity from a grammar.

That's why, elimination of Ambiguity from a grammar is undecidable.

eg1)

$G = \{ S \to AB \mid aab, A \to aA/a, B \to bB/b \}$

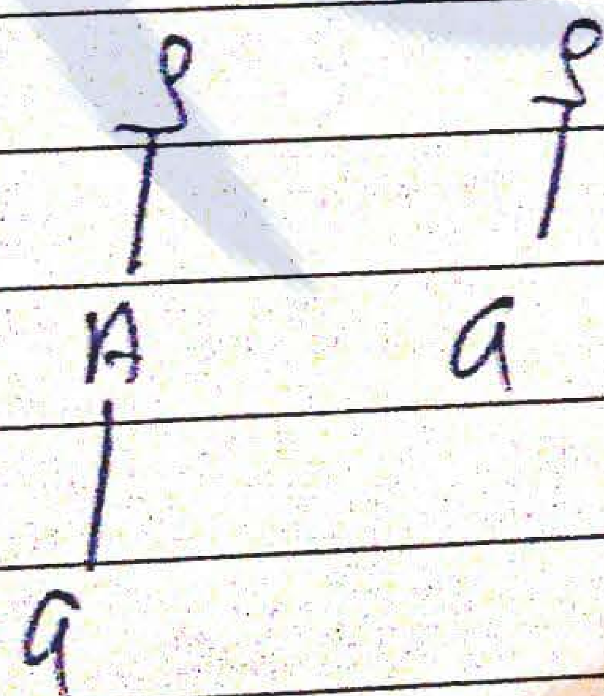
is ambiguous grammar

to eliminate,

$G' = \{ S \to AB, A \to aA/a, B \to bB/b \}$

is unambiguous grammar

eg2) $G = \{ S \to A/a, A \to a \}$



So, this is ambiguous

So,

$G' = \{ S \to a \}$

→ This is unambiguous

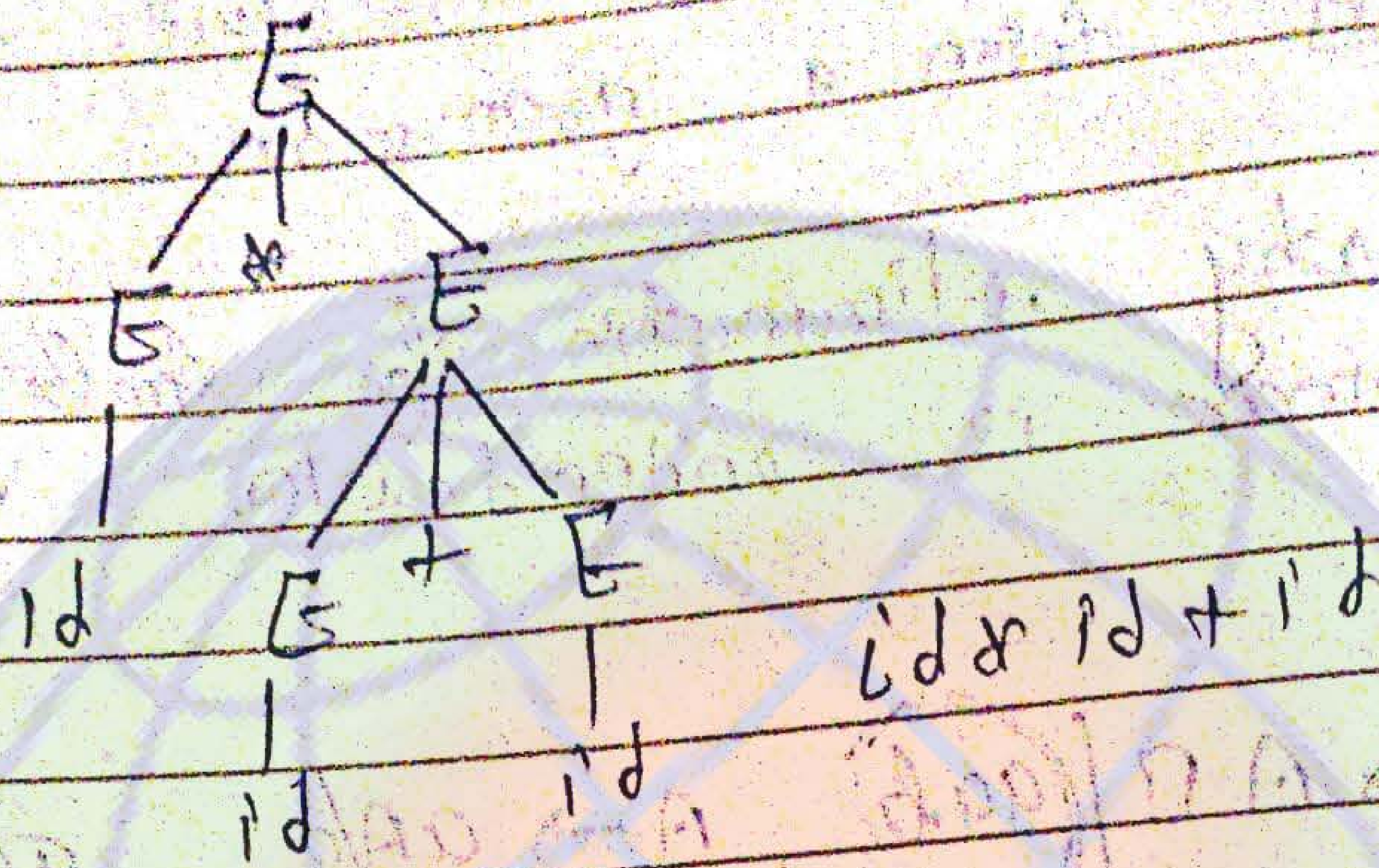
Scanned with CamScanner

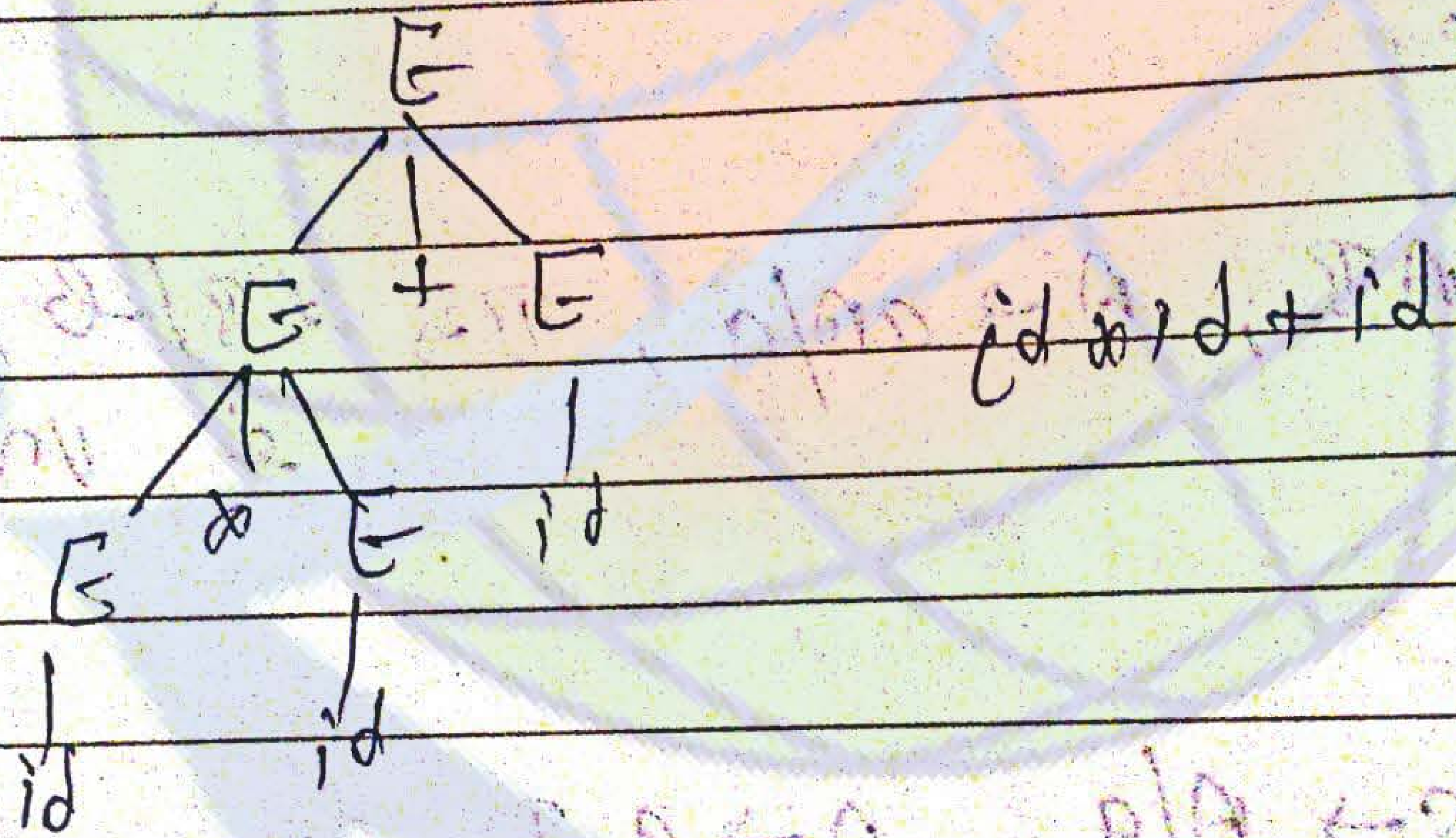
WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Note: कोई rule नहीं है, पूछे तो करें

Q3.) $G = \{E \rightarrow E+E \,/\, E*E \,/\, id\}$





so, the grammar is ambiguous

Now

$G = \{ E \rightarrow E+T \,/\, T$

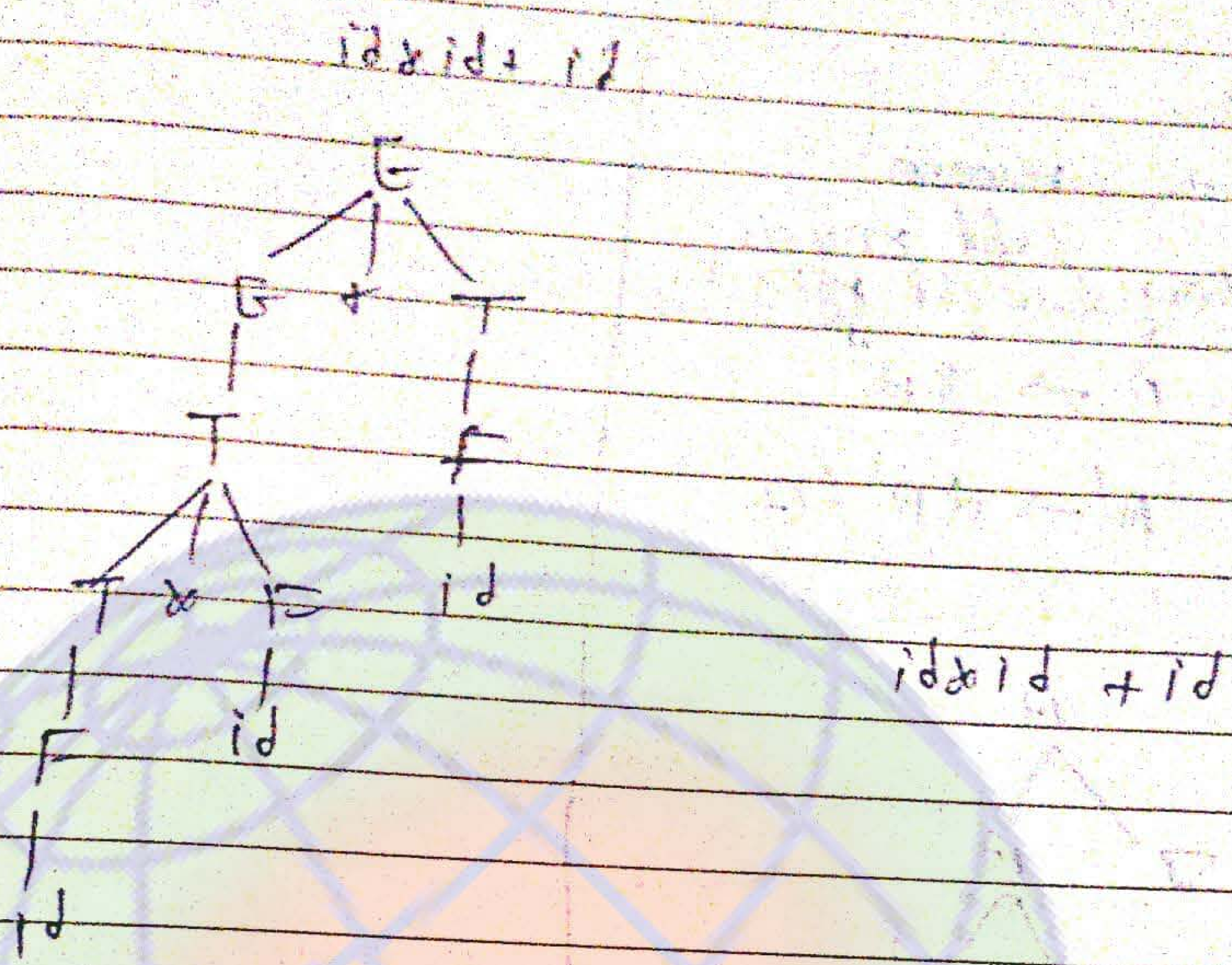
$T \rightarrow T*F \,/\, F$

$F \rightarrow id \}$ This is unambiguous grammar which gives all the string as above.

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM



Recursion Context free grammar

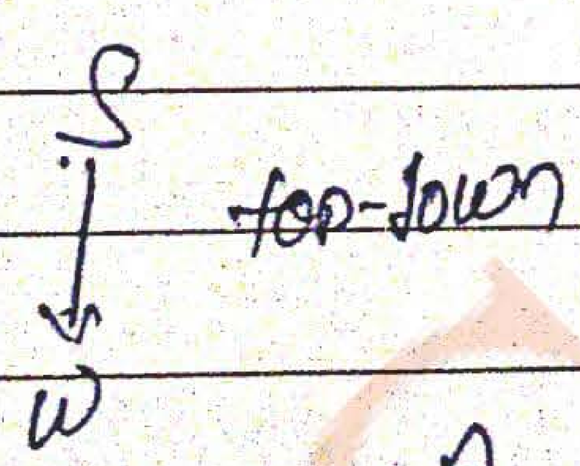
Left recursive CFG

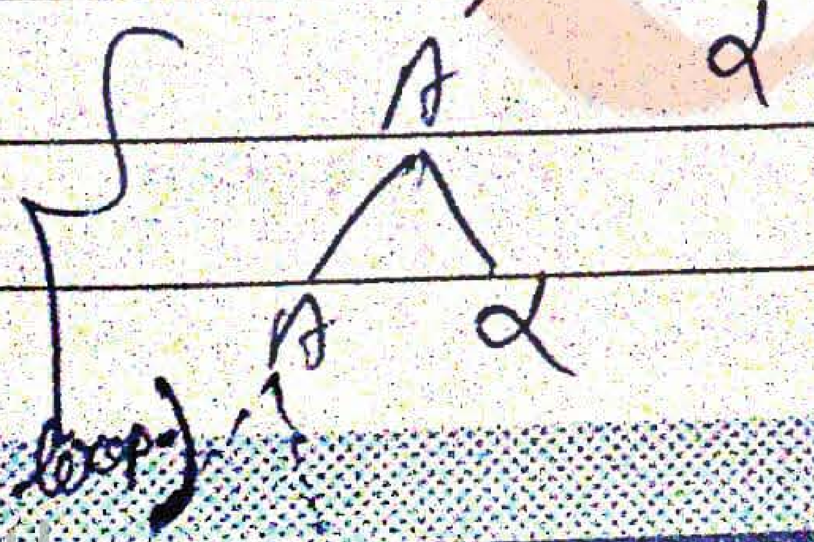
$A \rightarrow A\alpha / \beta$

Right recursive CFG

$A \rightarrow \alpha A / \beta$

$A \rightarrow A\alpha$
$\rightarrow A\alpha A$
$\rightarrow \beta\alpha\alpha$





the parser may not decide where to put β and go into infinite loop

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Page No.
Date

Eg:-
to remove left recursion
change to right recursion

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' / \epsilon$

-ves. derived

In the case of top-down parsing, if the grammar is left recursive, then the parser may fall into infinite loop. Therefore we must eliminate the left-recursion.

$S \rightarrow w$

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

R1

Page No.
Date

eg.) $G = \{ A \rightarrow A\alpha_1 \mid A\alpha_2 \ \ldots \ \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n \}$

R1

└ This is left recursive

So, to eliminate this

The formula:

$G: \{ A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \beta_3 A' \mid \ldots \mid \beta_n A'$

$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_n A' \mid \epsilon \}$

└ left recursion removed

Q.) eliminate the left recursion from the following grammar?

$G = \{ S \rightarrow Sa \mid Sb \mid a \mid b \}$

Sol:

$G = \{ S \rightarrow \underset{A\ \alpha_1}{Sa} \mid \underset{A\ \alpha_2}{Sb} \mid \underset{\beta_1}{a} \mid \underset{\beta_2}{b} \}$

$G' = \{ S \rightarrow aS' \mid bS'$

$S' \rightarrow aS' \mid bS' \mid \epsilon \}$

Q.) eliminate the left recursion from the

$G = \{ S \rightarrow AaS \mid b,\ A \rightarrow AbA \mid Sb\ S \mid a \}$

$G' = \{ S \rightarrow AaS \mid b,\ A \rightarrow \underset{A\ \alpha_1}{AbA} \mid \underset{A\ \alpha_2}{AaSb} \mid \underset{\beta_1}{bbS} \mid \underset{\beta_2}{a} \}$

$G'' = \{ S \rightarrow AaS \mid b,\ A \rightarrow bbSA' \mid aA',\ A' \rightarrow bAA' \mid aSbSA' \mid \epsilon \}$

Scanned with CamScanner

WWW.GRADESETTER.COM

★ Left factor:-

$G = \{A \rightarrow \alpha\beta_1 / \alpha\beta_2 / \text{---} / \alpha\beta_n\}$

— same prefix common,

Now, remove left ~~prefix if common~~ factor,

$G' = \{A \rightarrow \alpha A'$

$A' \rightarrow \beta_1 / \beta_2 / \text{---} / \beta_n\}$

Note The above was problem with top-down parser.

* $G = \{S \rightarrow \underset{\alpha\ \beta_1}{aAS} / \underset{\alpha\ \beta_2}{aAb} / b,\ A \rightarrow a\}$

~~$G' = \{S \rightarrow aA'$~~
~~$A' \rightarrow$~~

$G' = \{S \rightarrow aAS' / b,$

$S' \rightarrow S / b,\ A \rightarrow a\}$

* $G = \{S \rightarrow \underset{\alpha\ \beta_1}{01AS} / \underset{\alpha\ \beta_2}{011A} / \underset{\alpha\ \beta_3}{00},\ A \rightarrow 0A / 1\}$

$G' = \{S \rightarrow 0S',\ A \rightarrow 0A / 1$

$S' \rightarrow \underset{\alpha\ \beta_1}{1AS} / \underset{\alpha\ \beta_2}{11A} / 0$

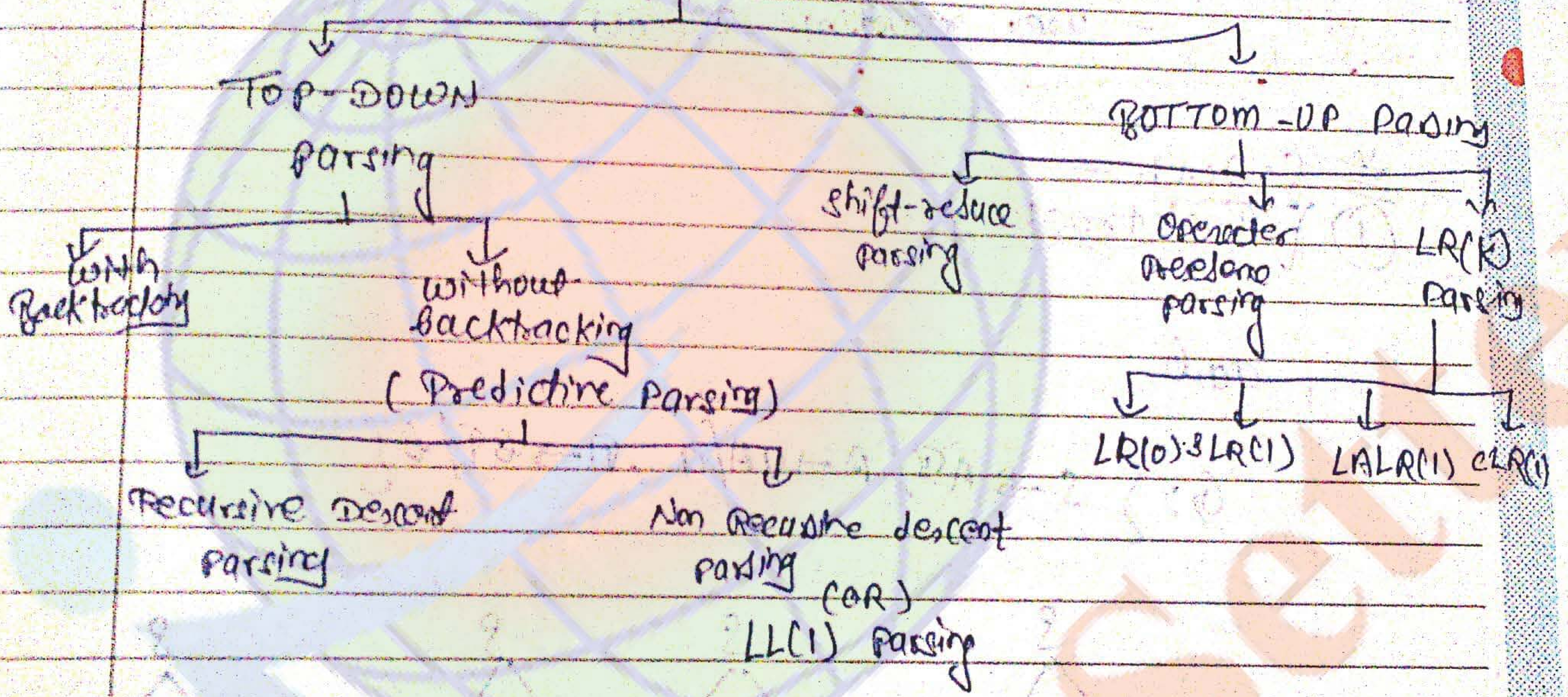
Scanned with CamScanner

WWW.GRADESETTER.COM

Page No.
Date

$G'' = \{ S \to 0S', A \to 0A/1,$
$S' \to 1S'/0$
$S'' \to AS/1A \}$

Parsing:-



① Top-down parsing : (from root to leaf)

- uses left most derivation (LMD)

$G = \{ S \to AB$
$A \to aA/a$
$B \to bB/b$

S
AB
aAB
aaB
aab

- uses derivation process.

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

(ii) Bottom-up parsing (from leaf to root)

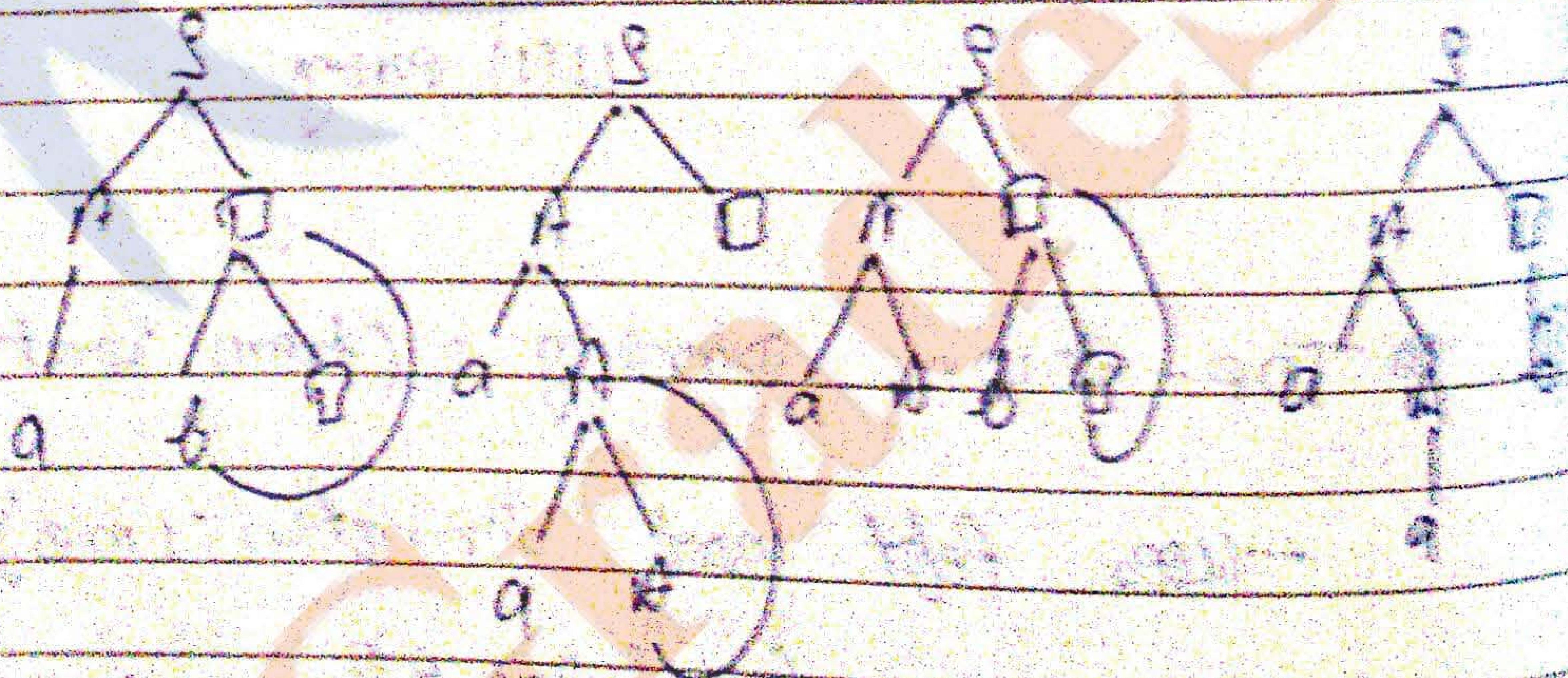
aab
aAb — uses right most
Ab derivation (RMD)
AB in reverse
S

— uses reduction process

① With-Backtracking:-

aab

G: {S → AB, A → aA/a, B → bB/b}



Backtracking a process of trying different productions untill it got the required string.
If the grammar is having too many productions, and if the string is also too

Scanned with CamScanner
WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

large, then the parser has to try so many productions.

because of these several tries the parser will become slow. that is why we dont prefer back tracking.

* FIRST set :-

First(A) - First of "TERM" i.e the set of

A → a a / a   first(A) = a

ab / a   first(A) = a

ab / C   a, C

aA / C   a, C

aA   a

A → aA / C

A → BC   first(A) = First(B) (B should not be ε)

B → bB / b   = {b}

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

- A → BC

B → b B / ε
First (B) = {b, ε}

First (A) = First (B)

= {b, ~~ε~~} ∪ First (C)

~~First~~ "FIRST" निकालने का तरीका

* V.V.I **Procedure:-**

1. First (a) = {a}

2.) If A → a ⇒ First (A) = {a}

3) If A → ε ⇒ First(A) = {ε}

4) If A → BC ⇒ First (A) = First(B) if First(B) does not contain ε.

R1 5) If A → BC ⇒ First (A) = First(B) − {ε} ∪ First(C)

मतलब — If B also have epsilon

Q) * Find the first of each non-terminal of the following grammar.

(1) S → ABC	First: {a, b}
A → aA / b	{a, b}
B → bB / ε	{b, ε}
C → cC / ε	{c, ε}

Note: ~~Always~~ Better go from the bottom

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

R1 (VVI)
R2

First(S) = {a, b}
First(A) = {a, b}
First(B) = {b, ϵ}
First(C) = {c, ϵ}

(Q2) G = { S → ABC
A → aA/ϵ
B → bB/c
C → cC/d }

First(S) = {a, b, c} — {a}∪{b}∪{c}
First(A) = {a, ϵ} — R1
First(B) = {b, c}
First(C) = {c, d}

3) G = { E → TE'
E' → +TE'/ϵ
T → FT'
T' → *FT'/ϵ
F → (E)/id }

First(E) = {(, id}
First(E') = {+, ϵ}
First(T) = {(, id}
First(T') = {*, ϵ}
First(F) = {(, id}

4) G = { S → ACB / CbB / Ba
A → da / BC
B → g/ϵ
C → h/ϵ }

First(S) = {d, g, h, b, a, ϵ} = {d, g, h, ϵ} ∪ {h, ϵ} ∪ {g, ϵ}
First(A) = {d, g, h, ϵ} = {g, ϵ} ∪ {h, ϵ}
First(B) = {g, ϵ}
First(C) = {h, ϵ}

R2

WWW.GRADESETTER.COM
Scanned with CamScanner

WWW.GRADESETTER.COM

Page No.
Date

Q(5)

$S \rightarrow AaB \,/\, bC$
$A \rightarrow BC \,/\, bA$
$B \rightarrow aB \,/\, \epsilon$
$C \rightarrow a \,/\, SA \}$

Sol^n

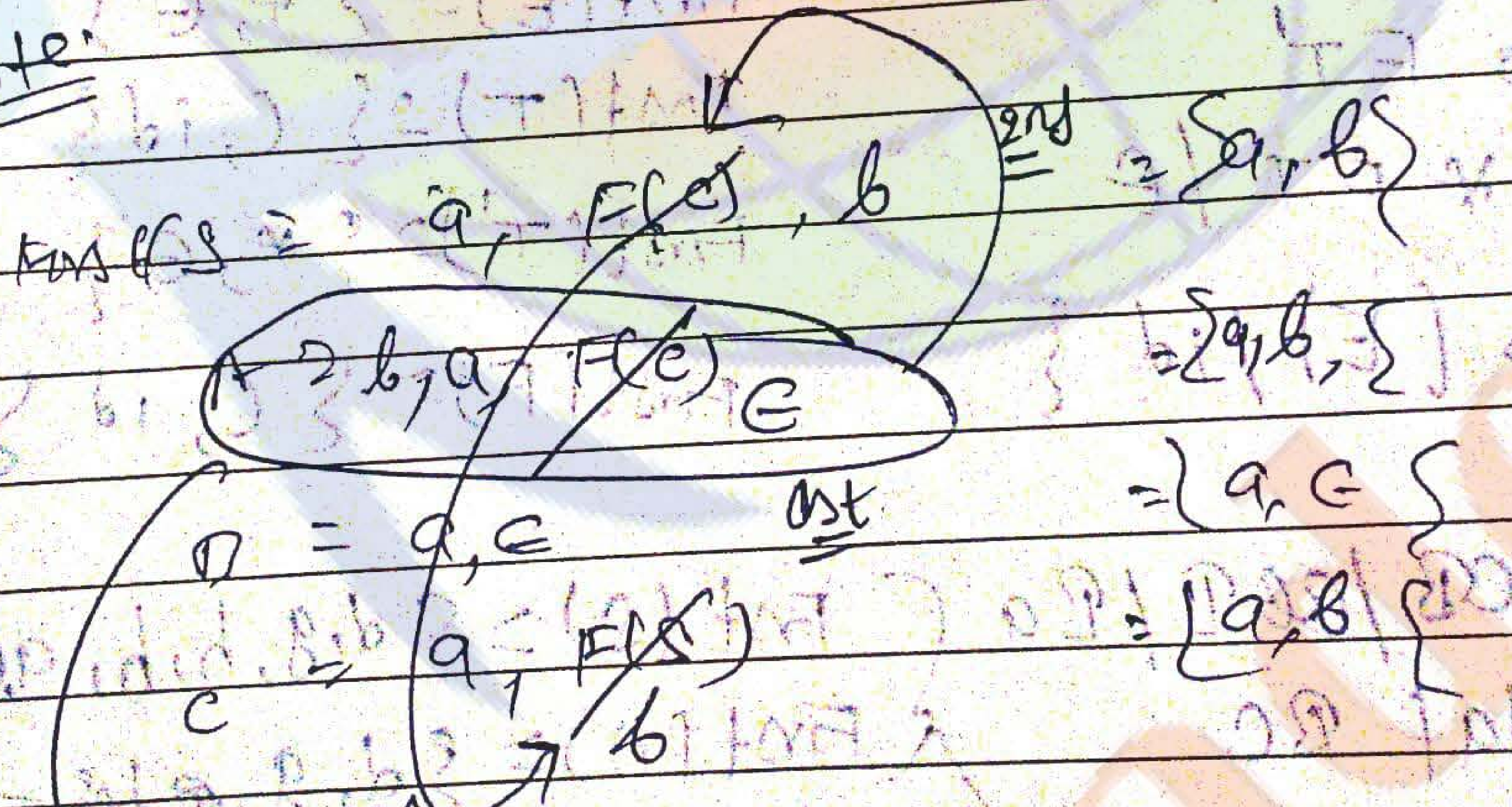
First (S) = {a, b} ✓

First (A) = {a, b} ∪ First...

First (B) = { a, ϵ }

First(C) = {a, b} ✓

Note:



Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Q) $G = \{ S \to abS \mid BA \mid \epsilon$

$A \to BC$

$B \to ab \mid Ac \mid \epsilon$

$C \to cC \mid d \}$

~~First~~
~~{a, b, c}~~
~~{ First(B) = {a, b, ε} c, d }~~
~~{a, First(A), ε}~~
~~{c, d}~~

[illegible]

$First(S) = \{a, b, \epsilon\}$

$First(A) = \{a, c, d\}$

$First(B) = \{a, c\} \cup First(A) = \{a, c, d, \epsilon\}$

$First(C) = \{c, d\}$

Q) Consider the following grammar

$S \to ABC$

Let the no. of elements in First(A), F(B), F(C) are 3, 4, 5 respectively. First of all these contains ε, and the remaining all symbols are different then the no. of elements in F(S) is

Sol

$S \to ABC$

$F(S) =$

$F(A) = \cancel{3}\ 2$

$F(B) = \cancel{4}\ 3$

$F(C) = \cancel{5}\ 4$

$\} + \epsilon = 10$ Ans

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Page No.
Date

Follow set:

$S \to ABCD$

Follow(A) = First

Follow of (A) is the set of terminals, that present immediately to the right side of (A), whenever (A) is on the right side of the arrow.

Procedure: "Follow निकालने का आसान तरीका"

1) Follow (S) = { \$ }

2) If $A \to \alpha B D \Rightarrow$ Follow(B) = First(D) if First(D) does not contain ϵ.

3) If $A \to \alpha B D \Rightarrow$ Follow B = First(D) − {ϵ} ∪ Follow(A)

4) If $A \to \alpha B \Rightarrow$ Follow(B) = Follow(A)

Q) Find the follow of each non-terminal of the following grammar.

① G = { $S \to AaAb \mid BbBa$
$A \to aA \mid \epsilon$
$B \to bB \mid \epsilon$ }

	First	Follow
S →	{ a, b, ~~\$~~ }	{ \$ }
A →	{ a, ϵ }	{ a, ~~b~~ }
B →	{ b, ϵ }	{ b, ~~a~~ } or { a, b }

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Page No.
Date

2) S → ABC
A → aA / b
B → bB / ε
C → cC / a

Follow(S) = { $ }
Follow(A) = { b, a, c }
Follow(B) = { c, a }
Follow(C) = { $ }

3) E → E+T
| T
T → T*F
| F
F → (E)
| id

Follow(E) = { +,), $ }
Follow(T) = { +, *,), $ }
Follow(F) = { +, *,), $ }

4) S → ACB / CbB / Ba
A → da / BC
B → g / ε
C → h / ε

Follow(S) = { $ }
Follow(A) = { h, g, $ }
Follow(B) = { $, a, h, g }
Follow(C) = { g, $, b, h }

{ g, ε } ∪ { $ }, { b }

{ h, ε } ∪ { g, ε } ∪ { $ }

{ $ } ∪ { a } ∪ { h, ε } ∪

Scanned with CamScanner
WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

write a lead sy...

uses LMD

Left to right scan of the input string

Construction of LL(1) Parsing table:-

*) we must consider every production of the form

$A \rightarrow \alpha$

and we proceed as follows

1) we write $A \rightarrow \alpha$ in

$M[A, x]$ where $x \in First(\alpha)$

2.) If $First(\alpha) = \{\epsilon\}$ then

write $A \rightarrow \alpha$ in $M[A, y]$

where $y \in Follow(A)$

3) If all the entries in the table are single then the grammer LL(1)

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

LL(1) :-

- L — Left to right scan of the input string
- L — uses LMD
- (1) — uses a head symbol of length 1

Construction of LL(1) Parsing table:-

*) we must consider every production of the form

$A \to \alpha$

and we proceed as follows

1) we write $A \to \alpha$ in

$M[A, x]$ where $x \in First(\alpha)$

2.) If $First(\alpha) = \{\epsilon\}$ then

write $A \to \alpha$ in $M[A, y]$

where $y \in Follow(A)$

3) If all the entries in the table are single then the grammar LL(1)

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

	a	b	$
S	S→AB	S→AB	Syntactical error
A	A→aA	A→B	Syntactical error
B	Syntactical error	B→bB	B→ε

Here, there is only single entry, so this is LL(1).

In the above table, all the entries are single.

Q) Construct LL(1) parsing table for the following grammar

G = { E → TE'
E' → +TE' / ε
T → FT'
T' → *FT' / ε
F → (E) / id }

	+	*	(	)	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→(E)		F→id	

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

$T' \to \epsilon$

$Follow(T') = Follow(T)$

$= First(E')$

$= \{+, \epsilon\} \cup \{), \$\}$

$= \{+,), \$\}$

In the above table, all the entries are single, together Therefore, the grammar is LL(1).

Stack	I/O Buffer	Action
$E	id + id * id $	Push by E → TE'
$ E'T'	id + id * id $	Push by T → FT'
$ E'T'F	id + id * id $	Push by F → id
$ E'T' id	id + id * id $	POP
$ E'T'	+ id * id $	Push by T' → ϵ
$ E'	+ id * id $	Push by E' → +TE'
$ E'T +	+ id * id $	POP
$ E'T	id * id $	Push by T → FT'
$ E'T'F	id * id $	Push by F → id
$ E'T' id	id * id $	POP
$ E'T'	* id $	Push by T' → *FT'
$ E'T'F *	* id $	POP
$ E'T'F	id $	Push by F → id
$ E'T' id	id $	POP

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

$T' \to \epsilon$

$Follow(T) = Follow(T')$

$= First(E')$

$= \{+, \epsilon\} \cup \{), \$\}$

$= \{+,), \$\}$

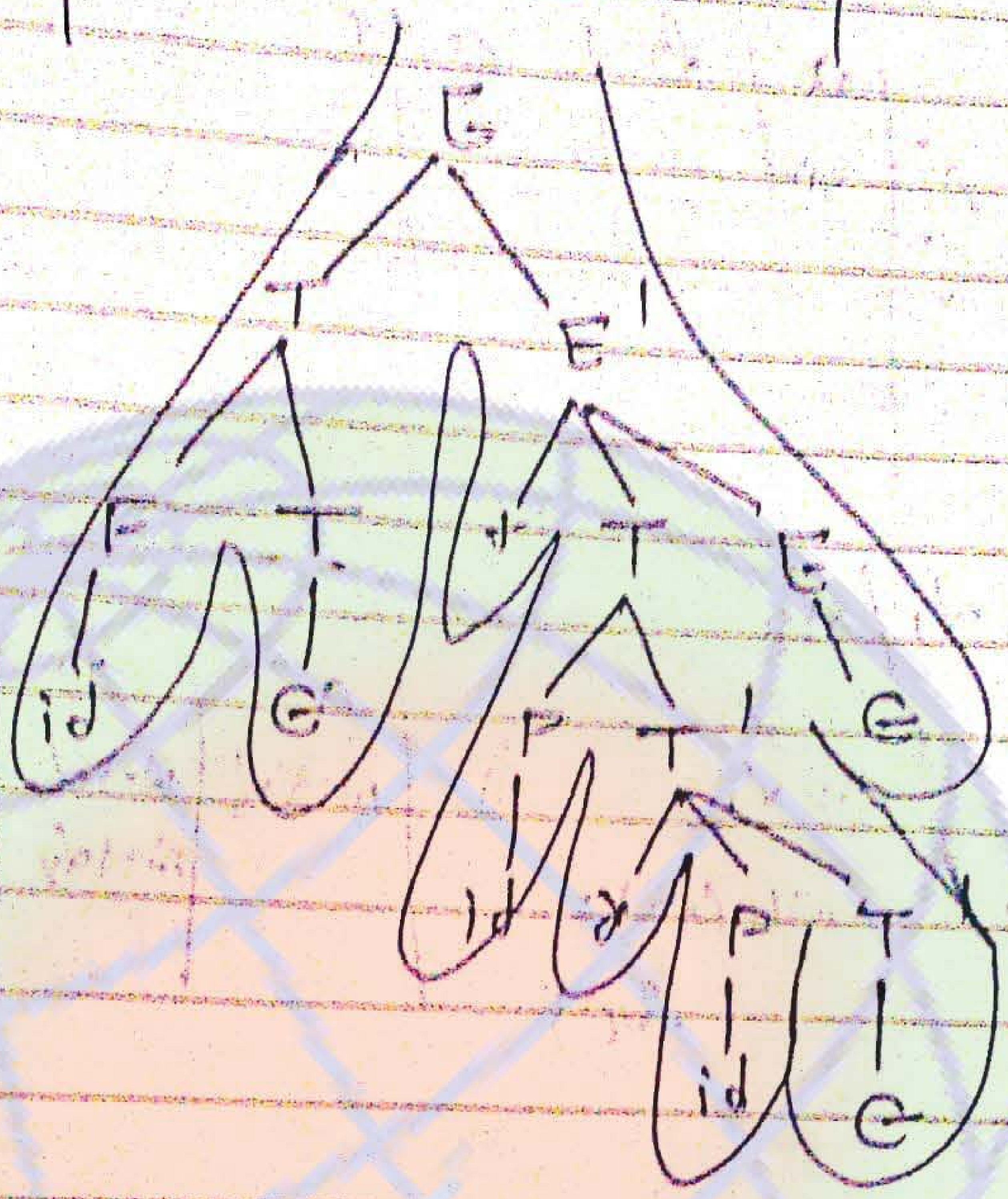
In the above table, all the entries are single.
Therefore the grammar is LL(1).

Stack	i/o Buffer	Action
$E	id + id * id $	Push by E → TE'
$E'T'	id + id * id $	Push by T → FT'
$E'T'F	id + id * id $	Push by F → id
$E'T'id	id + id * id $	POP
$E'T'	+ id * id $	Push by T' → ε
$E'	+ id * id $	Push by E' → +TE'
$E'T+	+ id * id $	POP
$E'T	id * id $	Push by T → FT'
$E'T'F	id * id $	Push by F → id
$E'T'id	id * id $	POP
$E'T'	* id $	Push by T' → *FT'
$E'T'F*	* id $	POP
$E'T'F	id $	Push by F → id
$E'T'id	id $	POP

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

$E'T'$	$	Push by $T \to \epsilon$
E'	$	Push by $E' \to \epsilon$
$	$	Accept



Properties of LL(1) grammar:-

1) A left recursive grammar is not LL(1).

$\{E \to E+E/id\}$

↳ not LL(1) because left recursive

2) A left factor grammar is not LL(1)

↳ two or more prefix

eg.

$S \to A B / A S$ → Not LL(1)

3) An Ambigius grammar is not LL(1)

4) Every LL(1) grammar is Unambigius.

but,

every Unambigius grammar is not LL(1)

Scanned with CamScanner

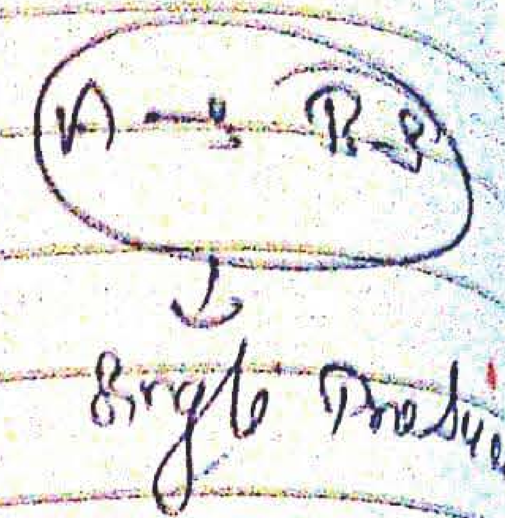
WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

5) Every single production grammar is LL(1).

eg.

	a	b	c	$
A				
B				



Imp. 6)

$A \rightarrow \alpha_1 / \alpha_2$

$A \rightarrow \alpha_1$ $\quad$ $A \rightarrow \alpha_2$

$First(\alpha_1) = \{a, b\}$ $\quad$ $First(\alpha_2) = \{b, c\}$

	a	b	c	$
A	$A \rightarrow \alpha_1$	$A \rightarrow \alpha_1$, $A \rightarrow \alpha_2$	$A \rightarrow \alpha_2$	

If any productions are of the form

$A \rightarrow \alpha_1 / \alpha_2$ then

If $First(\alpha_1) \cap First(\alpha_2) \neq \phi$

$\Rightarrow$ Not LL(1)

Note:

If any production are of the form

$A \rightarrow \alpha_1 / \alpha_2 / \alpha_3 / \ldots / \alpha_n$

then

If First of any two have common elements then it is not LL(1).

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Imp: If any productions are of the form

$A \to \alpha \mid \epsilon$

then

if $First(\alpha) \cap Follow(A) \neq \phi$

$\Rightarrow$ Not LL(1)

Q) $G = \{S \to Aa, A \to a\}$

Sol) $S \to A \mid a$

$First(A) \cap First(a)$

$= \{a\} \cap \{a\}$

$= \{a\}$

$= a$

$=$ Not empty

so not LL(1)

Q) $G = \{S \to aA \mid bB \mid a$

$A \to \epsilon$

$B \to \epsilon\}$

$\cap \neq 0$

so Not LL(1)

so, Ambiguous.

Sol) Not LL(1)

$First(aA) \cap First(a)$

$\{a\} \cap \{a\}$

$\neq \phi$

so, Not LL(1)

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Q3.) G = { S → ABC

A → aA / ε ,

B → bB / ε

C → cC / ε }

Sol LL(1)

First (aA) ∩ Follow(A)

= {a} ∩ {b, c, $}

= ϕ

(No common

so LL(1) ✓

First (bB) ∩ Follow(B)

= {b} ∩ {c, $}

= ϕ

First (cC) ∩ Follow(C)

= {c} ∩ {$}

= ϕ

Q4) [illegible]

G = {

Sol

Q5) check

not

G

Sol

Fo

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Q4) ~~soooo~~

$G = \{ S \to aS / bA,\ A \to aSA / bA / \epsilon \}$

Sol/n

First(aS) ∩ First(bA)

{a} ∩ {b}

= φ

= First(aSA) ∩ First(bA) ∩ Follow(A)

= {a} ∩ {b} ∩ {a, b, $}

≠ φ

∴ Not LL(1)

Q5) check weather the following grammar is LL(1) or not and also find the entries of

M [S', e] & M [E, a]

in the corresponding ~~pp~~ parsing table.

$G = \{ S \to iEtSS' / a,\ S' \to eS / \epsilon,\ E \to b \}$

Sol/n

First(iEtSS') ∩ First(a)

i ∩ a

= φ

First(eS) ∩ Follow(S')

e ∩ {e, $}

≠ φ

Follow(S) = First(S')

= {e, ~~ε~~} ∪ {$}

= {e, $}

So

Not LL(1)

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Q4) ~~soooo~~

$G = \{ S \to aS / bA, \; A \to aSA / bA / \epsilon \}$

sol^n

First(aS) ∩ First(bA)

{a} ∩ {b}

$= \phi$

= First(aSA) ∩ First(bA) ∩ Follow(A)

= {a} ∩ {b} ∩ {a, b, $}

$\neq \phi$

= Not LL(1)

Q5) check weather the following grammar is LL(1) or not and also find the entries of

m [S', e] & m [E, a]

in the corresponding ~~to~~ parsing table.

$G = \{ S \to iEtSS' / a, \; S' \to eS / \epsilon, \; E \to b \}$

sol^n

First(iEtSS') ∩ First(a)

i ∩ a

$= \phi$

First(eS) ∩ Follow(S')

e ∩ {e, $}

$\neq \phi$

Follow(S) = First(S')

= {e, ~~ε~~} ∪ {$}

= {e, $}

So

Not LL(1)

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
	[illegible]	

m [[illegible]] = { [illegible] , [illegible] }

m [[illegible]] = φ ← [illegible]

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Bottom-up Parsing:-

Page No. ____ Date ____

Handle:- A handle is a substring of a string that matched with any of the right side of the productions.

then that handle will be reduces by its left side of the production.

$G = \{ E \rightarrow E+E \mid E*E \mid id \}$

Handle Prooning

Right Sentential Form	Handle	Production
id + id * id	id	E → id
E + id * id	id	E → id
E + E * id	E+E	E → E+E
E * id	id	E → id
E * E	E*E	E → E*E
E		

Handle Prooning:- Bottom-up parsing is a process of finding the handles, and using them in the reduction to get the start symbol.
This entire process is called handle Prooning

Shift Reduce Parsing:

In this parsing technique, we use two datastructures.

1) stack - uses to store the symbols of the grammar.

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Bottom-up Parsing:-

Page No. ______
Date ______

Handle:- A handle is a substring of a string that matches with any of the right side of the productions.

then that handle will be reduces by its left side of the production.

G = { E → E+E | E*E | id }

Handle Prooning

Right Sentential Form	Handle	Production
id + id * id	id	E → id
E + id * id	id	E → id
E + E * id	E+E	E → E+E
E * id	id	E → id
E * E	E*E	E → E*E
E		

Handle Prooning:- Bottom-up parsing is a process of finding the handles, and using them in the reduction to get the start symbol.

This entire process is called handle Prooning.

Shift Reduce ~~Technique~~ Parsing:

In this parsing technique, we use two datastructures.

1) stack - uses to store the symbols of the grammar.

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Page No.
Date

- The initial configuration of the stack is | $ |

2) Input buffer - uses for storing the input string

- The initial configuration of the input buffer is

| w | $ |

In this technique we perform four techniques:
   a) shift
   b) Reduce
   c) Accept
   d) Error

Q) Consider the following grammar and pass the input string id+id*id using shift reduce parsing

$E \to E+E \mid E*E \mid id$

stack	i/p Buffer	Action
$	id+id*id $	shift
$id	+id*id $	Reduce by E→id
$E	+id*id $	shift
$E+	id*id $	shift
$E+id	*id $	Reduce by E→id
$E+E	*id $	shift (Not reduce)

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Page No. ____
Date ____

$E+E*	id $	Shift
$E+E*id	$	Reduce by E → id
$E+E*E	$	Reduce by E → E*E
$ E+E	$	Reduce by E → E+E
$E	$	Accept

Shift = 5
Reduce = 5
Accept = 1

Total = 11

Operator Grammar :-

A Grammar is said to be operator grammar, if it satisfy the following two cond.

1) No production should contain adjacent non-terminals on the right side of the arrow.
2) There should not be any null production

eg1) G = { E → E + E / E * E / id } is an operator grammar.

eg2) G = { S → SAS / a, A → + / - / * / / } is not operator grammar

↓

G = { S → S+S / S-S / S*S / S/S / a } is an operator grammar.

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Consider the following grammar and priority relations of the following grammar

$G = \{E \rightarrow E+E / E*E / id\}$

	id	*	+	$
id	—	⋗	⋗	⋗
*	⋖	⋗	⋗	⋗
+	⋖	⋖	⋗	⋗
$	⋖	⋖	⋖	—

Note: ⋖ this dot is known as priority relation.

* Operator Precedence Parsing:-

- Construction of operator precedence parsing table -
  - Let x is the top most symbol in the stack and y is the present input symbol in the input buffer, then we proceed as follows.

1) If $x \lessdot y$ then perform shift operation
2) If $x \gtrdot y$ then there will be a handle in the stack then perform reduce operation with a proper production.

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Page No. ______
Date ______

Q) Consider the following grammar

$G = \{E \to E - E / E * E / id\}$

id - id * id

and parse the input string using operator presedence parsing.

Stack		i/o buffer	Action
$	<·	id - id * id $	Shift
$ id	·>	- id * id $	Reduce by E → id
$ E	<·	- id * id $	Shift
$ E -	<·	id * id $	Shift
$ E - id	·>	* id $	Reduce E → id
$ E - E	<·	* id $	Shift
$ E - E *	<·	id $	Shift
$ E - E * id	·>	$	Reduce E → id
$ E - E * E	·>	$	Reduce E → E * E
$ E - E	·>	$	" E → E - E
$ E		$	Accept.

* $G = \{E \to E + E / E * E / id\}$

f \ g	id	*	+	$
id	-	·>	·>	·>
*	<·	·>	·>	·>
+	<·	<·	·>	·>
$	<·	<·	<·	-

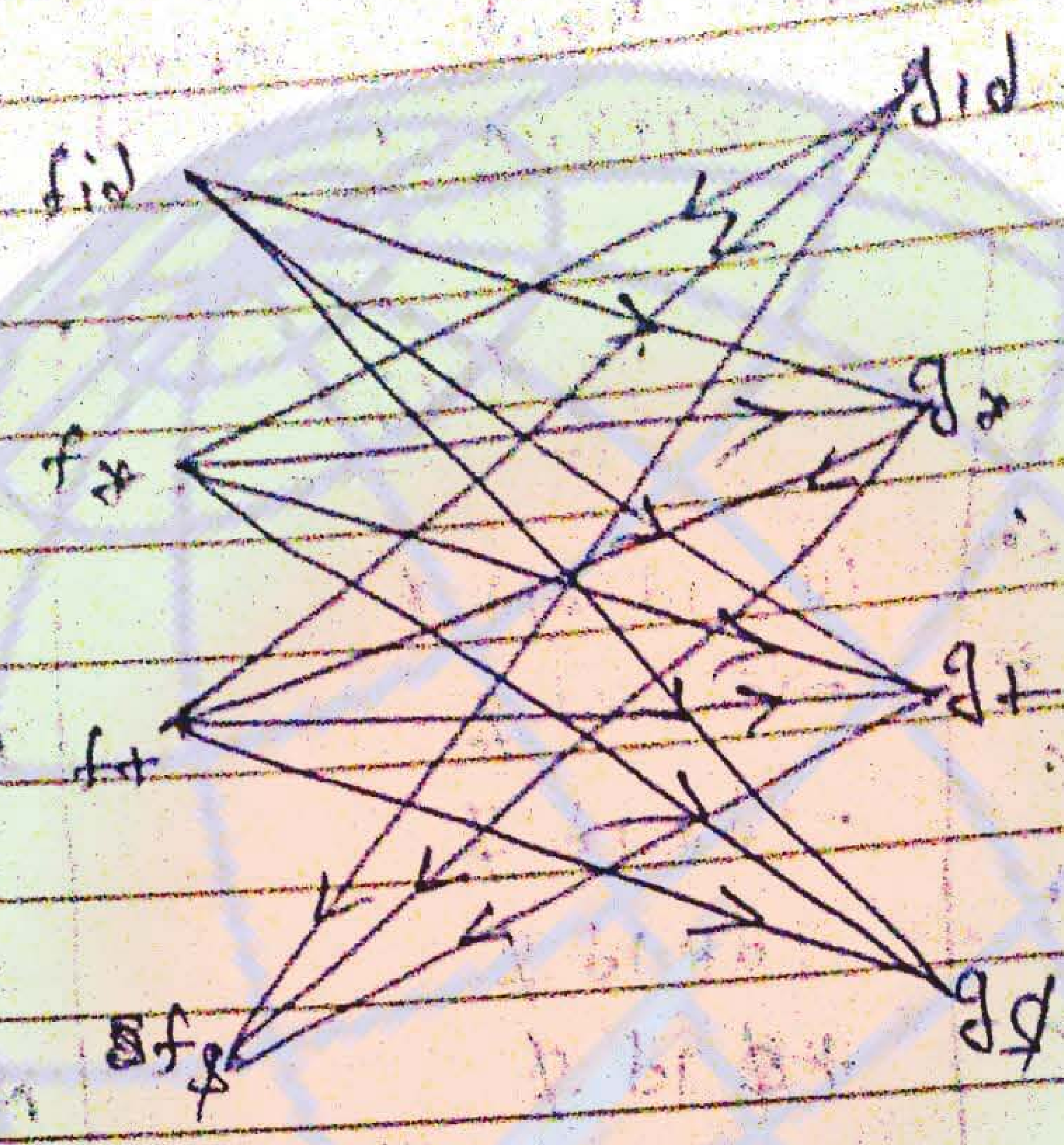
Scanned with CamScanner

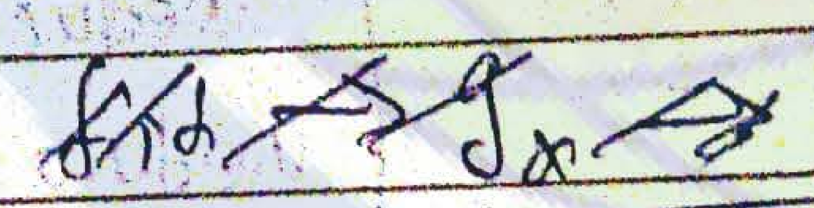
WWW.GRADESETTER.COM

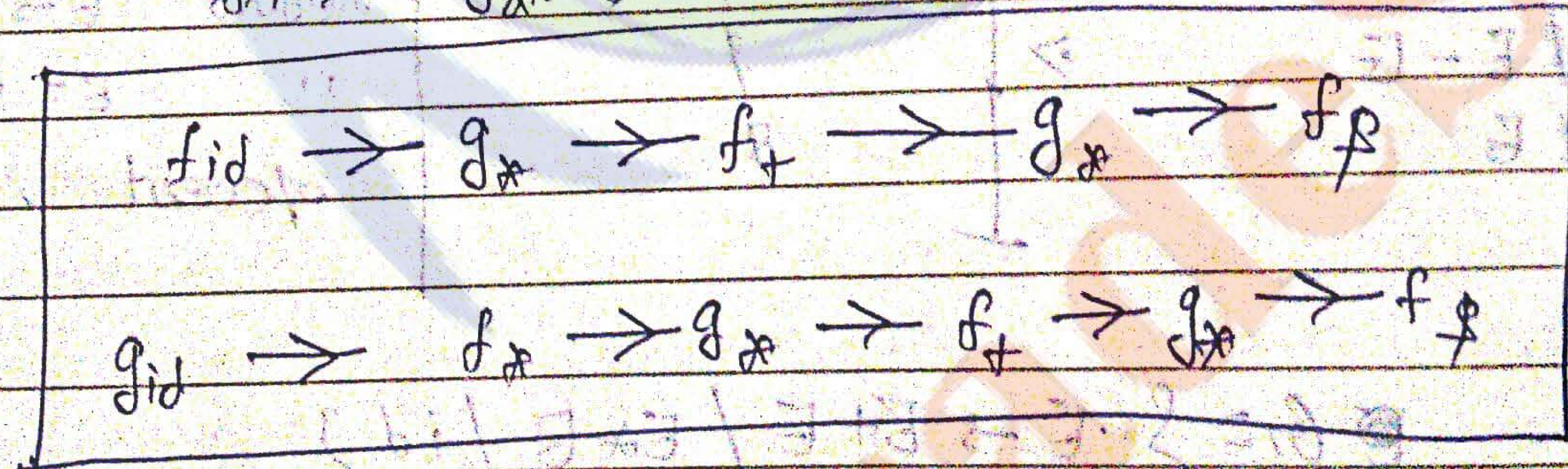
WWW.GRADESETTER.COM

The size of the operator relation table is $O(n^2)$

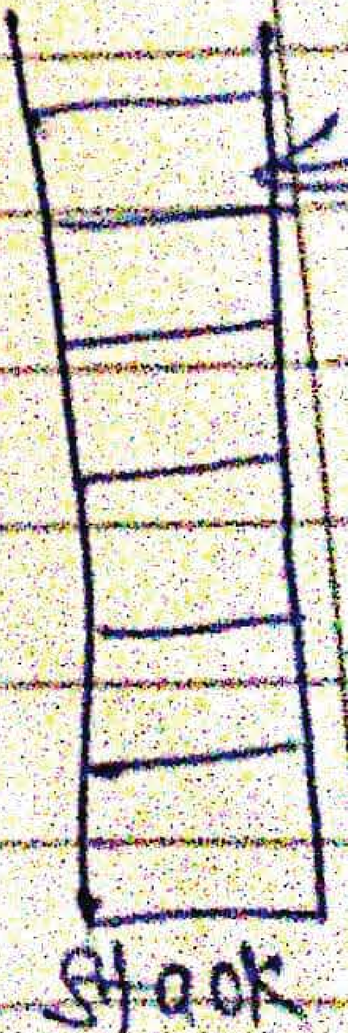
where n is number of operators.







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



Scanned with CamScanner

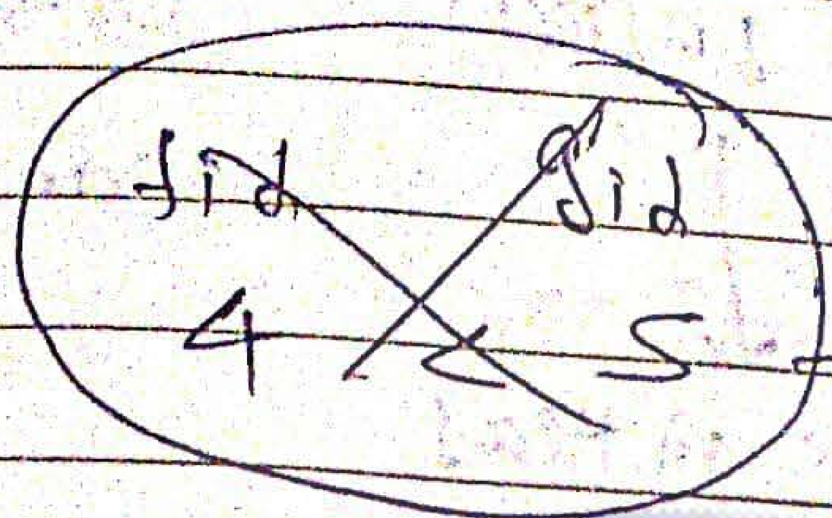
WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

eg.)

f_x g_x
4 > 3

f_x g_x
2 < 3

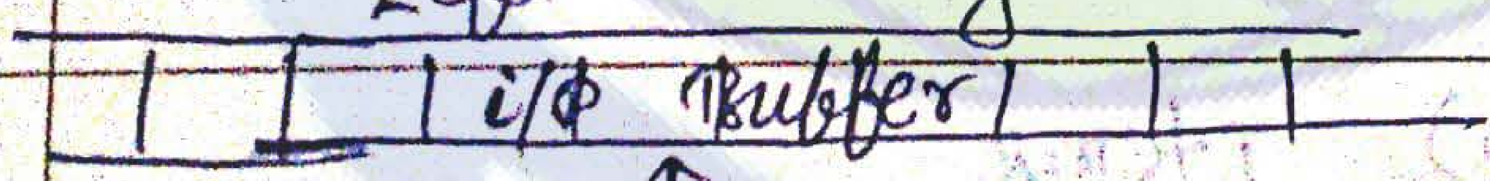


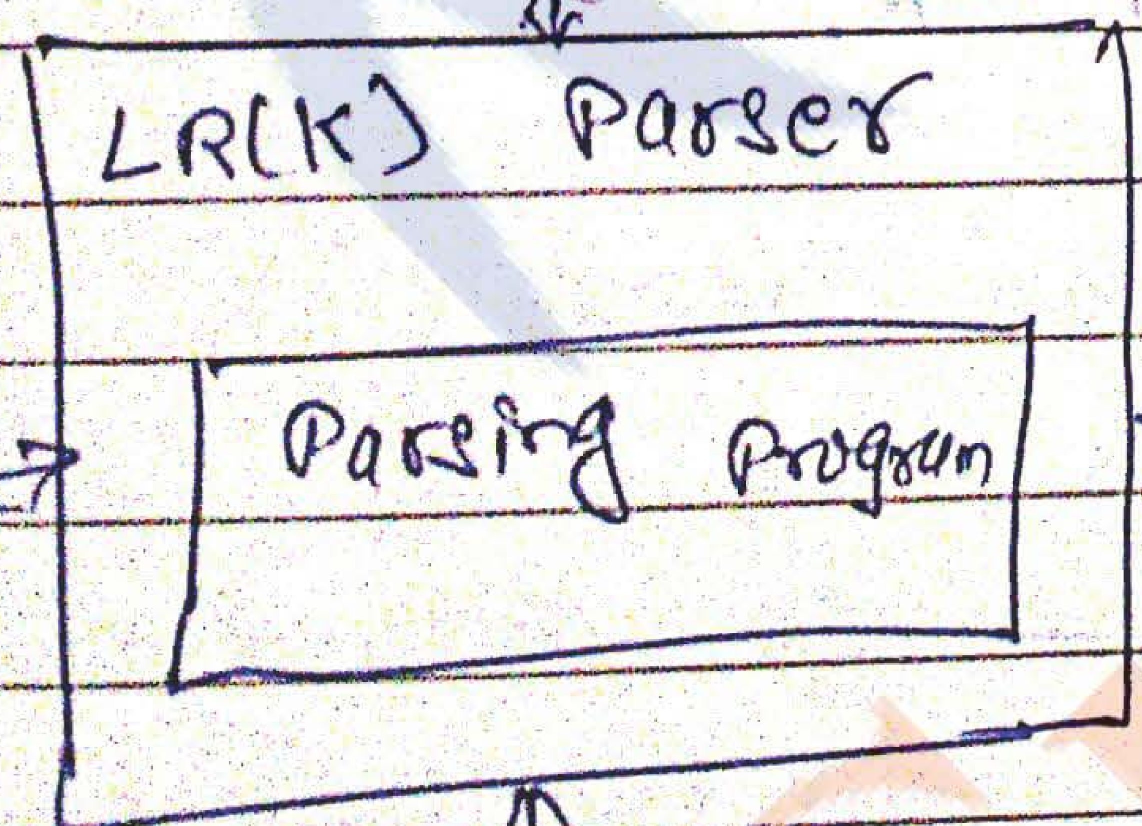
→ This is Unwanted relation

Now the size of the table is O(2n)

LR(k)

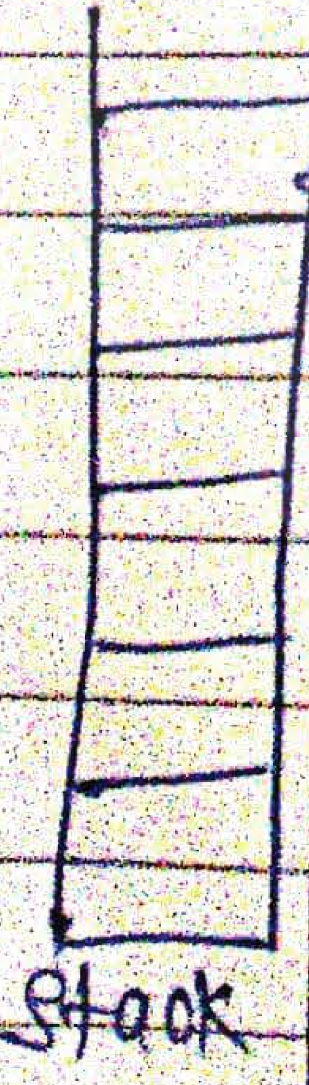
- k → Look a head symbol of length 'k'.
- R → uses RMD
- L → Left to right scan of the i/p string





→ O/P

Block diagram of LR(k) Parser



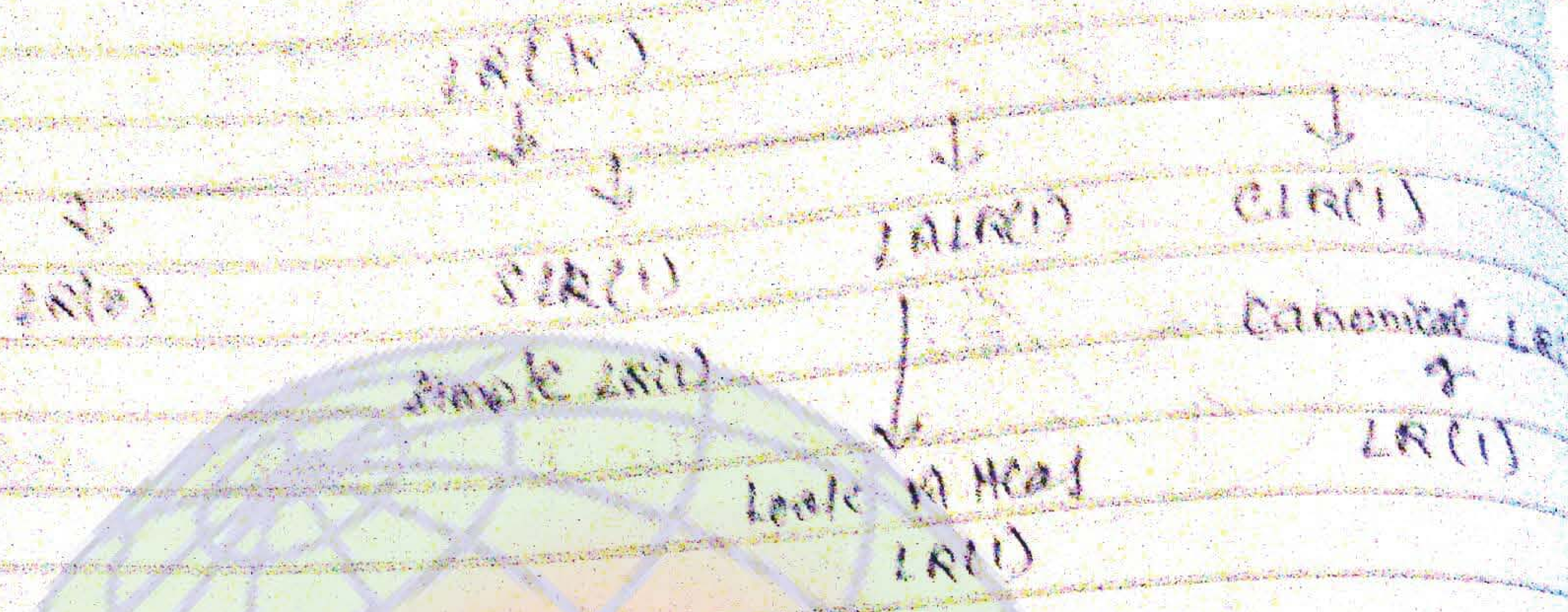
State	Action	GOTO
0		
1		
n		

Parsing table

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM



* SLR(1) Parsing

- SLR(1) has three steps

1. Construct of Canonical items (or) LR(0) items
2. " " parsing table
3. Parsing the i/p string

1. Construction of LR(0) items.

S → .xyz

S → x.yz

S → xy.z

S → xyz.

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Page No.
Date

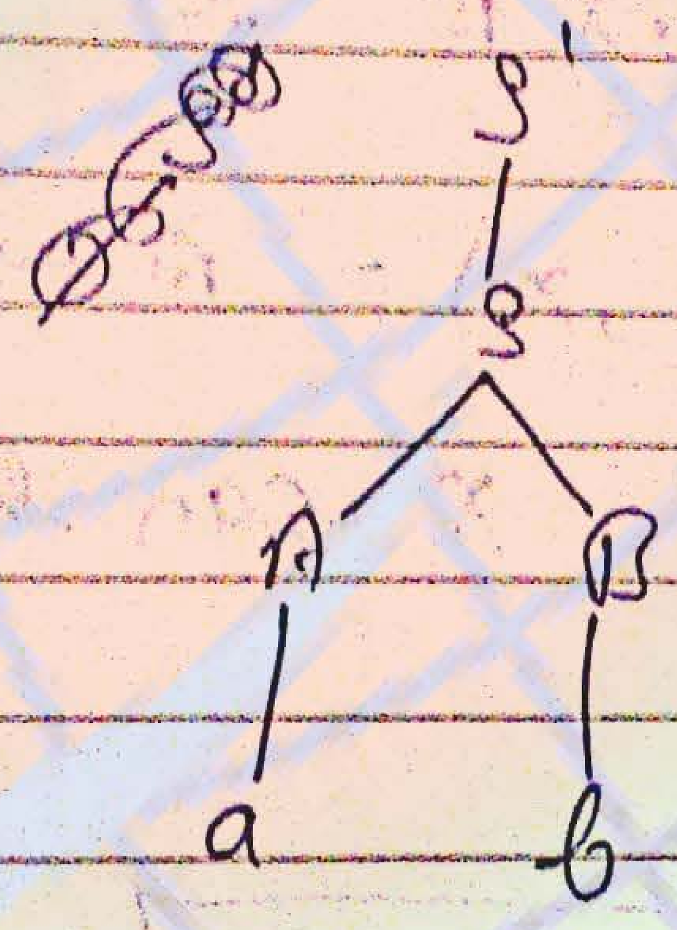
Augmented grammar:- Augmented grammar is a new grammar, with the new start symbol 'S'' such that

$S' \rightarrow S$ as a new production

The purpose of this production, the parser will get terminated, after successful parsing of the input string.

$G = \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\}$

$G' = \{S' \rightarrow S, S \rightarrow AB, A \rightarrow a, B \rightarrow b\}$



Construction of LR(0) items:-

Constructs on LR(0) items has two parts-

(i) Closure operation

(ii) Goto operation.

1) Construction of closure operation-

a) Initially add $S' \rightarrow \bullet S$

$G = \{S \rightarrow AB / a$

$A \rightarrow c$

$B \rightarrow b\}$

$S \rightarrow \bullet AB$

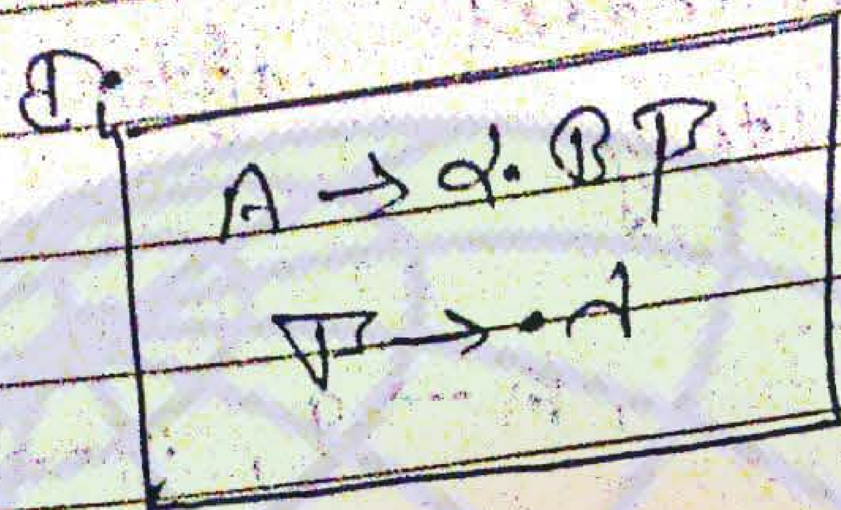
$A \rightarrow \bullet c$

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

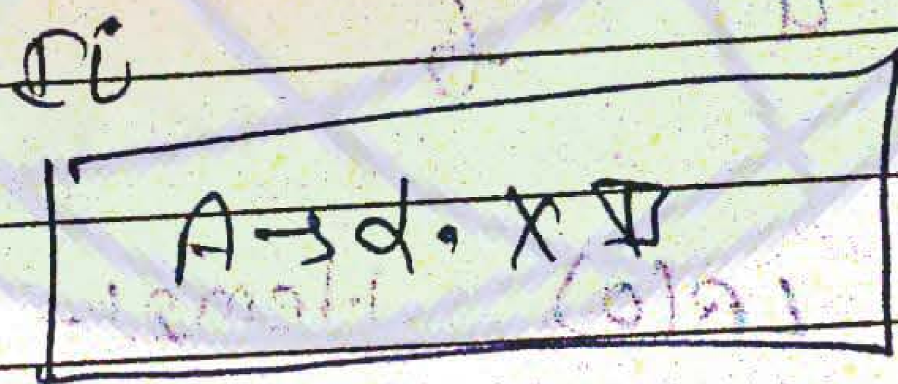
b) If $A \rightarrow \alpha \cdot B\beta$ is in I_i and if $B \rightarrow \gamma$ is a production, then add $B \rightarrow \cdot\gamma$ in I_i



ii) Goto operation —
If $A \rightarrow \alpha \cdot X\beta$ is in I_i, then

$$Goto(I_i, X) = A \rightarrow \alpha X \cdot \beta$$

Here X can be terminal or non terminal



$$Goto(I_i, X) = A \rightarrow \alpha X \cdot \beta$$

Q) Construct SLR(1) parsing table for the following grammar.

$$G = \{E \rightarrow E+T \mid T,\ T \rightarrow T*F \mid F,\ F \rightarrow (E) \mid id\}$$

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

I_0

$E' \to \cdot E$
$E \to \cdot E+T$
$E \to \cdot T$
$T \to \cdot T*F$
$T \to \cdot F$
$F \to \cdot(E)$
$F \to \cdot id$

I_1 Goto (I_0, E)

$E' \to E \cdot$
$E \to E \cdot + T$

I_2 Goto (I_0, T)

$E \to T \cdot$
$T \to T \cdot * F$

I_6 Goto $(I_1, +)$

$E \to E + \cdot T$
$T \to \cdot T * F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot id$

I_7: Goto $(I_2, *)$

$T \to T * \cdot F$
$F \to \cdot (E)$
$F \to \cdot id$

I_4 (Goto $(I_0, ( )$

$F \to (\cdot E)$
$E \to \cdot E+T$
$E \to \cdot T$
$T \to \cdot T*F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot id$

I_3 Goto (I_0, F)

$T \to F \cdot$

I_5 Goto (I_0, id)

$F \to id \cdot$

I_8: Goto (I_4, E)

$F \to (E \cdot)$
$E \to E \cdot + T$

Goto $(I_4, T) = I_2$
Goto $(I_4, F) = I_3$
Goto $(I_4, ( ) = I_4$
Goto $(I_4, id) = I_5$

I_9: Goto (I_6, T)

$E \to E + T \cdot$
$T \to T \cdot * F$

Goto $(I_6, F) = I_3$

~~$T \to F \cdot$~~

I_{10} Goto (I_7, F)

$T \to T * F \cdot$

Goto $(I_6, ( ) = I_4$
Goto $(I_6, id) = I_5$

Goto $(I_7, ( ) = I_4$
Goto $(I_7, id) = I_5$

I_{11} $(I_8,))$

$F \to (E) \cdot$

Goto $(I_8, +) = I_6$
Goto $(I_9, *) = I_7$

Scanned with CamScanner
WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Now,
Construction of SLR(1) parsing table-
It has two parts
1) ACTION
2) GOTO

1) Action - Action part of the table will be filled as follows-

$A \to \alpha . a \beta$
$A \to \alpha . B \beta$
$A \to \alpha a \beta .$

a) If $A \to \alpha . a \beta$ is in I_i, for every symbol after that we write goto

If $A \to \alpha . a \beta$

and If $Goto(I_i, a) = I_j$

then, set $Action(i, a) = S_j$

b) If $A \to \alpha .$ is in I_i
then
set $Action[i, x] = $ Reduce by $A \to \alpha$
where $x \in Follow(A)$

Here $A \to \alpha$ must not be an augmented production

c) If $S' \to S .$ is in I_i, then
set $Action[i, \$] = $ Accept

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

2) Goto

a) If [$A \rightarrow \alpha . X\beta$] is in I_i
and if

Goto (I_i, X) = I_j then

set Goto (i, X) = j

1) $E \rightarrow E + T$
2) $E \rightarrow T$
3) $T \rightarrow T * F$
4) $T \rightarrow F$
5) $F \rightarrow (E)$
6) $F \rightarrow id$

Follow(E) = { +,), $ }

Follow(T) = { *, +,), $ }

Follow(F) = { *, +,), $ }

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Page No.
Date

	ACTION						GOTO		
States	+	*	(	)	id	$	E	T	F
0			S4		S5		1	2	3
1	S6					Accept			
2	R2	S7		R2		R2			
3	R4	R4		R4		R4			
4			S4		S5		8	2	3
5	R6	R6		R6		R6			
6			S4		S5			9	3
7			S4		S5				10
8	S6			S11					
9	R1	S7		R1		R1			
10	R3	R3		R3		R3			
11	R5	R5		R5		R5			

(E' → E.) → Augmented (Accept)

∴ (1, $) Accept

(E → T.) → Reduce

R6

F → id.

[5, $] : R6

In the above table, all the entries are single therefore the grammer is SLR(1)

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Conflict निकालने का आसान तरीका

Page No. ____
Date ____

Conflicts in SLR(1)

i) Shift - Reduce
ii) Reduce - Reduce

i) Shift - reduce (S/R) conflict :-

I_i

$A \to \alpha . a \beta$
$B \to \gamma .$

	a	b	c	$
i				

If $\{a\} \cap Follow(B) \neq \phi$

$= S/R$ conflict

$=$ Not SLR(1)

ii) Reduce - reduce conflict (R/R)

I_i

$A \to \alpha .$
$B \to \alpha .$

If

$Follow(A) \cap Follow(B) \neq \phi$

$= R/R$ conflict

$=$ not SLR(1)

Q) Trick

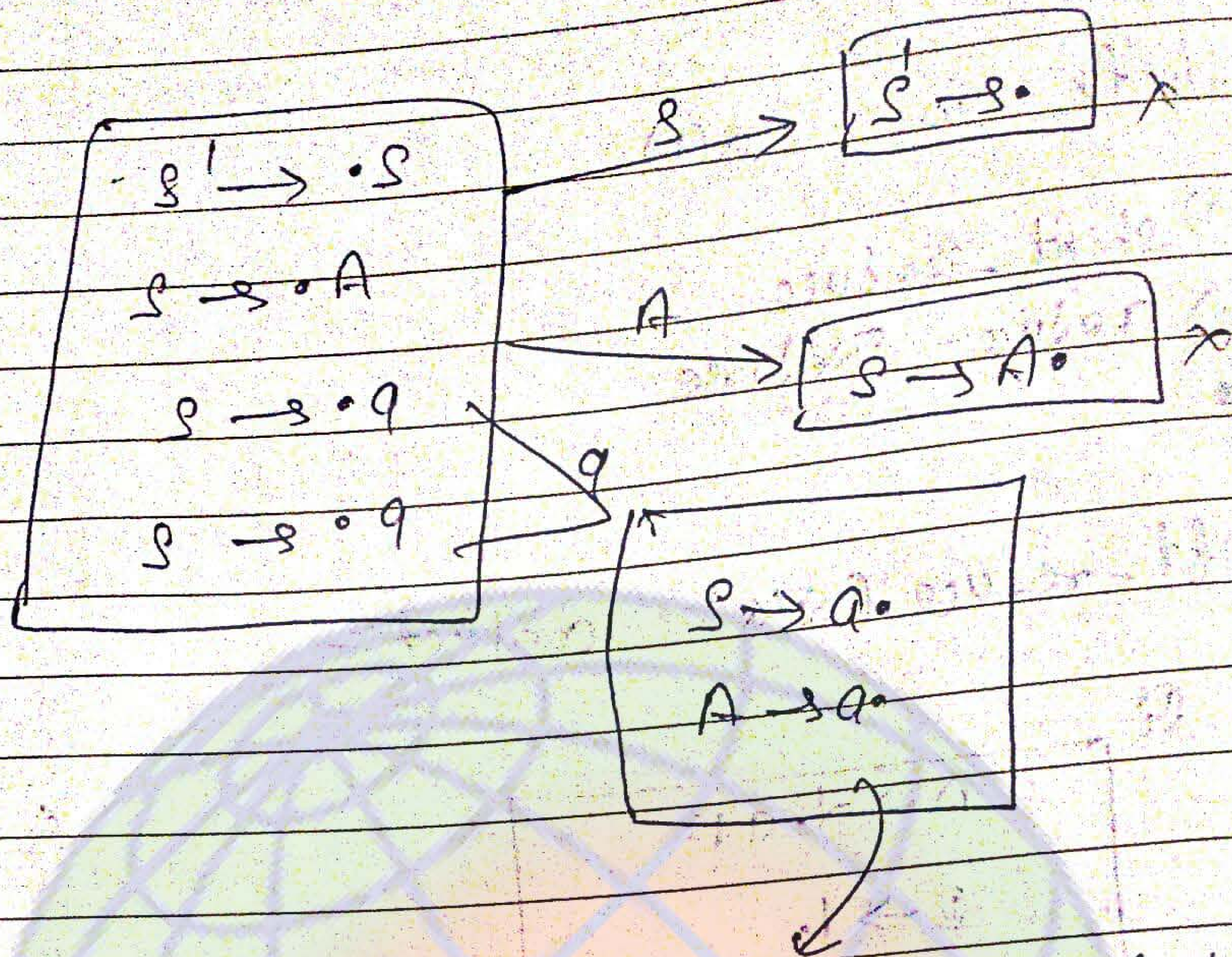
Check weather the following grammar is SLR(1) or not

1) $G = \{S \to A/a,\ A \to a\}$

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM



Follow(S) ∩ Follow(A)

= {$} ∩ {$}

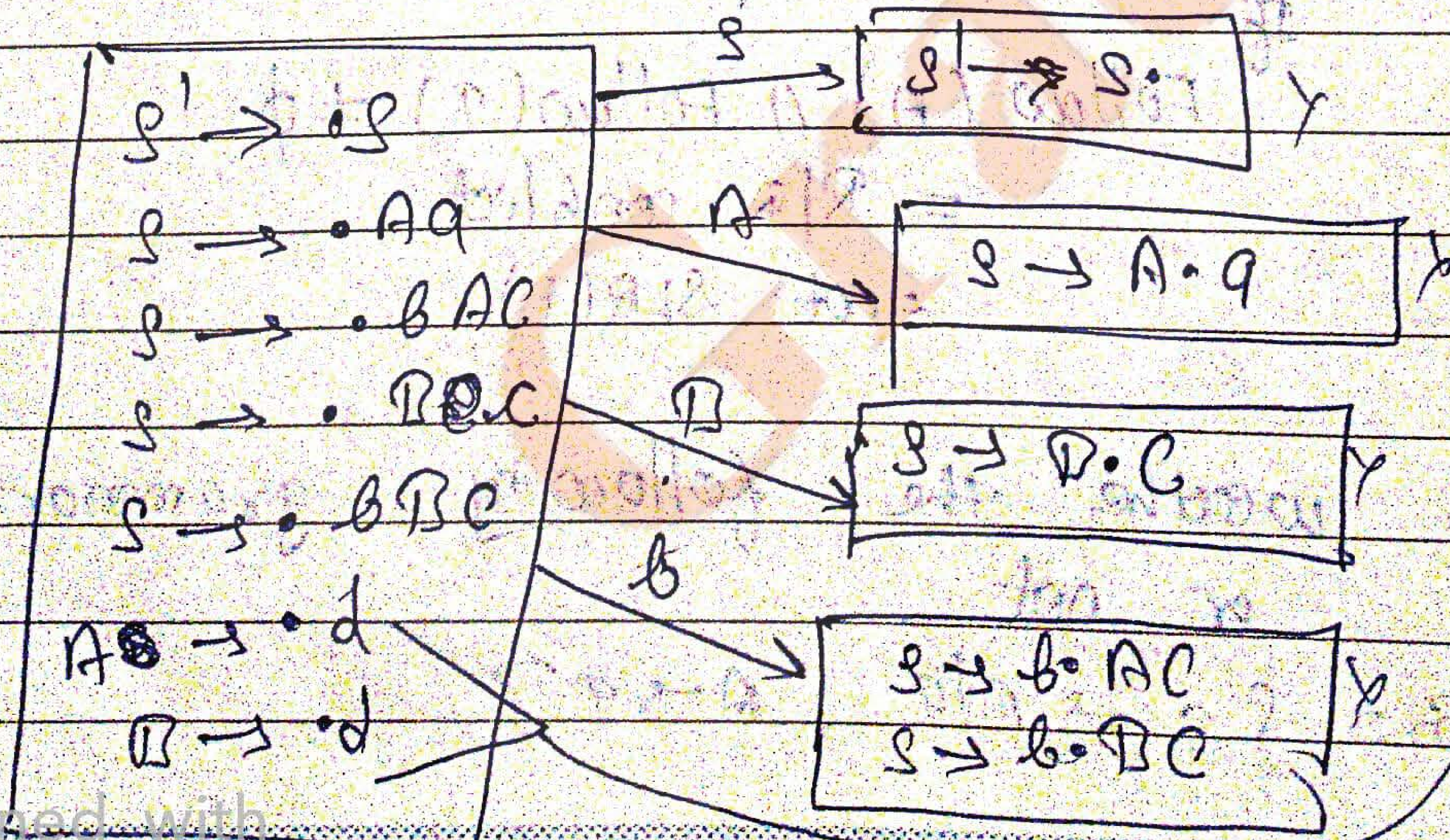
= {$}

= ϕ

= R/R conflict

= Not SLR(1)

2) G={S → Aa / bAc / Bc / bBc , A → d, B → d}

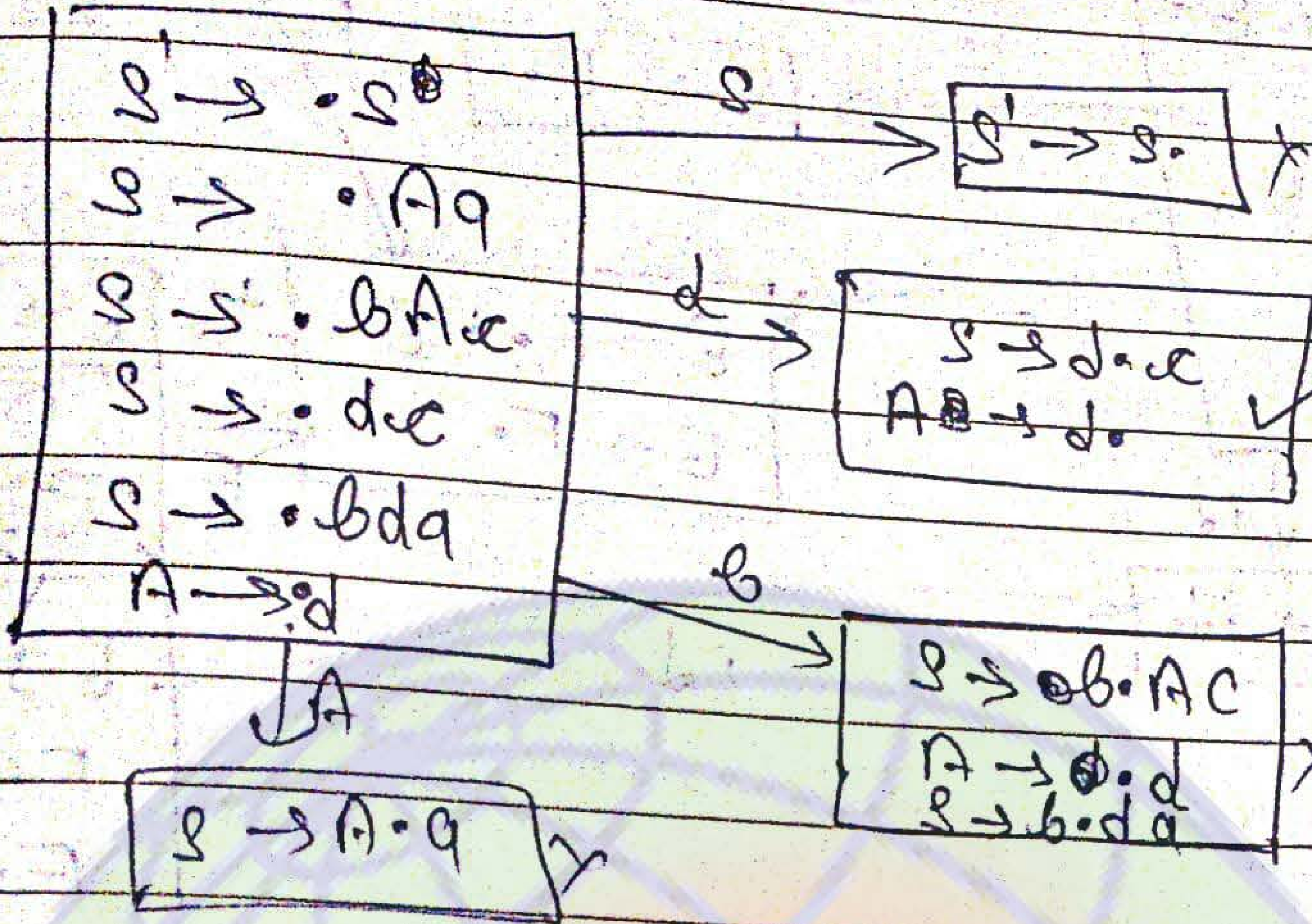


WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

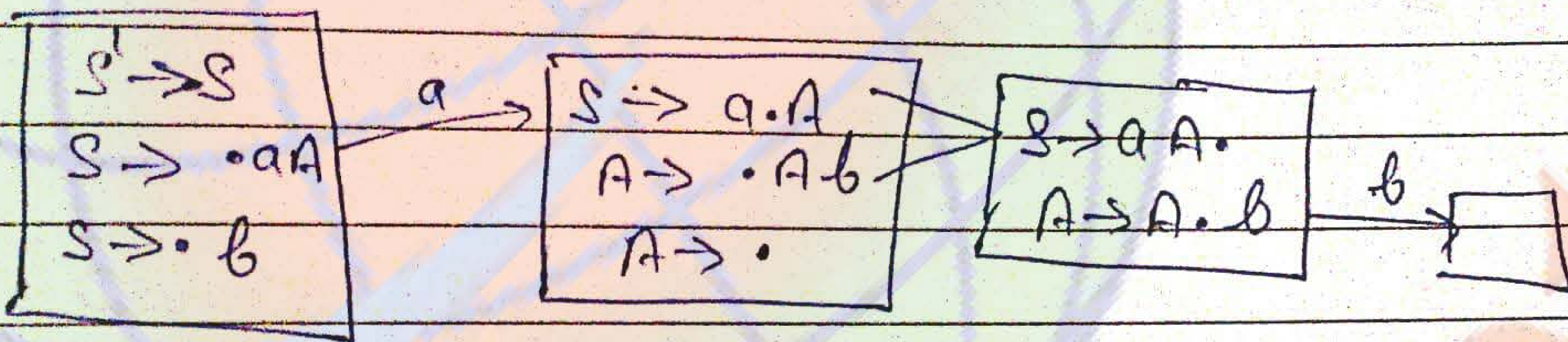
Page No. ____
Date ____

3) $G = \{ S \to Aa \mid bAc \mid dc \mid bda \ , \ A \to d \}$



⇒ $\{c\} \cap Follow(A)$
$\{c\} \cap \{a, c\} \neq \phi$
→ S/R conflict
⇒ Not SLR

4) $G = \{ S \to aA \mid b \ , \ A \to Ab \mid \epsilon \}$



$= \{b\} \cap Follow(S)$

$= \{b\} \cap \{\$\}$

$= \emptyset$

a) Consider the following grammar under LR(0) items of the grammer

$G = \{ S \to S * E \mid E, \ E \to F + E \mid F \ , \ F \to id \}$

1) $S \to S * \cdot E$ 2) $E \to F \cdot + E$ 3) $E \to F + \cdot E$

Given the items above, which two of them will be same set of canonical set of items of the grammar.

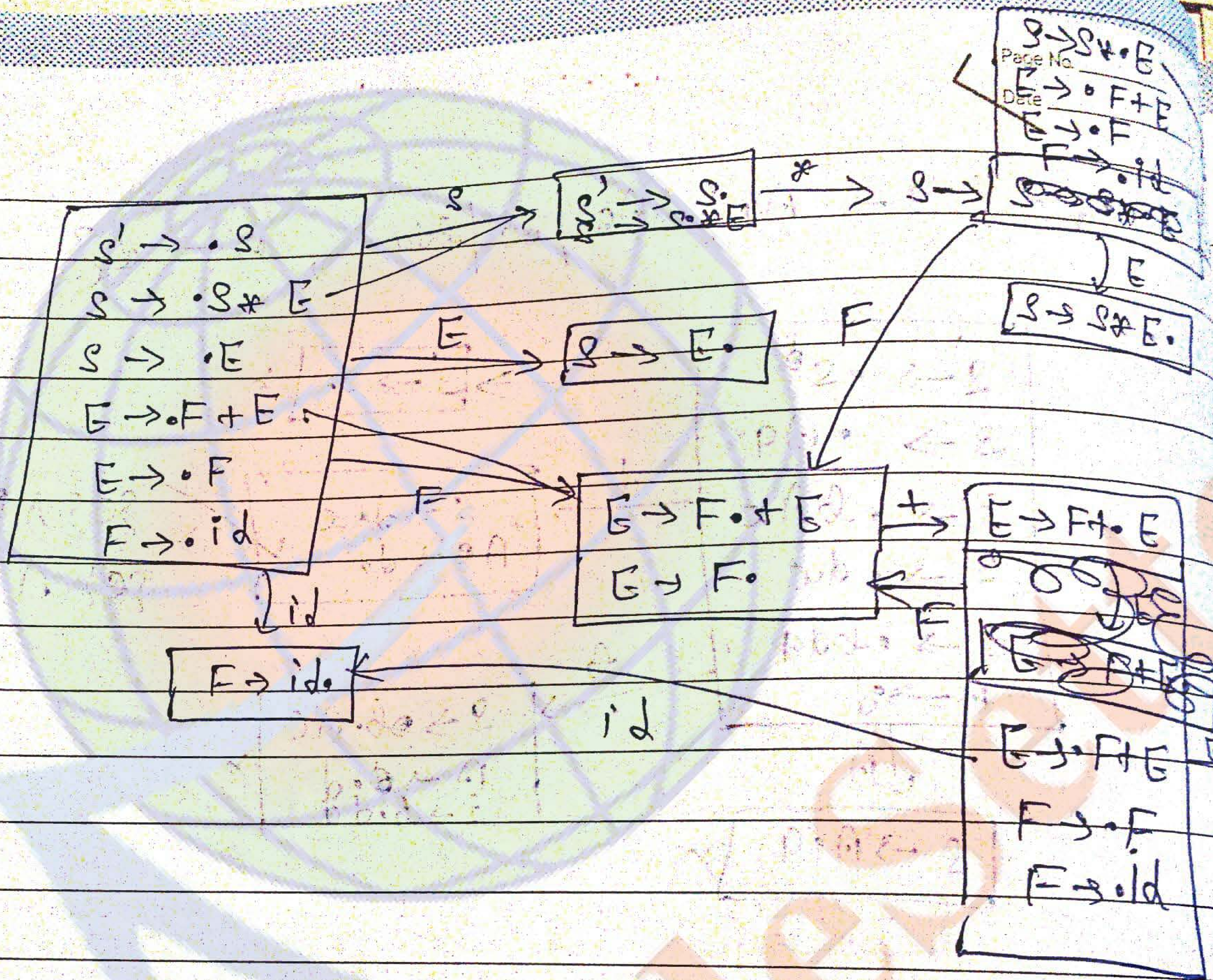
a) 1&2 b) 2&3 c) 1&3 d) none

So, R/R conflict
Not SLR(1)

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM



WWW.GRADESETTER.COM

Scanned with CamScanner

WWW.GRADESETTER.COM

★ LR(0) Parsing:

LR(0) parser has two parts:

a) Construction of LR(0) items
b) Con n of parsing table

1) Construction of LR(0) items:
It is same as construct of LR(0) items in SLR(1).

2) Construction of parsing table-
It is also same as SLR(1), except that, If

S_i [$A \to \alpha.$] is in S_i, then write reduce by $A \to \alpha$ in entire i'th row of action part of the table.

Conflicts:-

1) Shift-reduce
2) Reduce-Reduce

① Shift-reduce conflict

S_i [$A \to \alpha . a \beta$; $B \to \alpha .$]

	a	b	e	f
i	S_j / R_i	S_j / R_i	R_i	R_i

(conflict)

⇒ S/R conflict
⇒ Not LR(0)

2) Reduce-Reduce conflict

S_i [$A \to \alpha.$; $B \to \alpha.$]

	a	b	c	$
i	R_1/R_2	R_1/R_2	R_1/R_2	R_1/R_2

⇒ R/R conflict ⇒ Not LR(0)

Scanned with CamScanner

WWW.GRADESETTER.COM

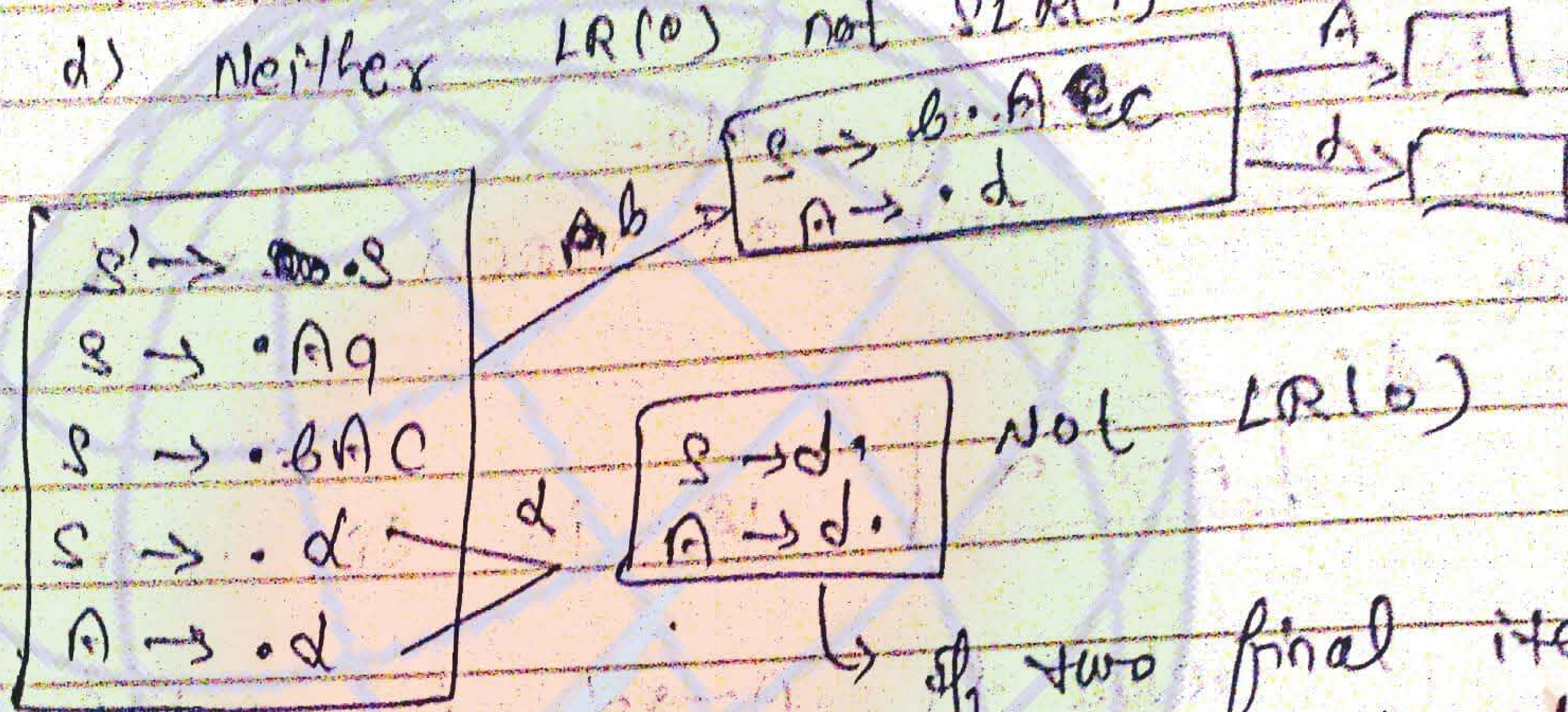
WWW.GRADESETTER.COM

Q) The following grammar is

$G_1 = \{ S \to Aa \,/\, bAc \,/\, d, \quad A \to d \}$

a) LR(0) but not SLR(1)
b) SLR(1) but not LR(0)
c) both LR(0) & SLR(1)
d) Neither LR(0) nor SLR(1)

Sol^n



↳ If two final items then R/R conflict in LR(0)
⇒ Not LR(0)

Now,

Follow(S) ∩ Follow(A)

$= \{\$\} \cap \{a, c\}$

$= \phi$

∴ The grammar is SLR(1)

Note:

LR(0) ⊂ SLR(1)

Every LR(0) grammar is also SLR(1), but every SLR(1) is not LR(0)

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

★ CLR(1) or LR(1) :-

R1

It has two parts

1) construction of Canonical set of items or LR(1) items — R1
2) construction of parsing table

① construction of LR(1) items.

It has two parts :-

a) closure operations
b) Goto operation

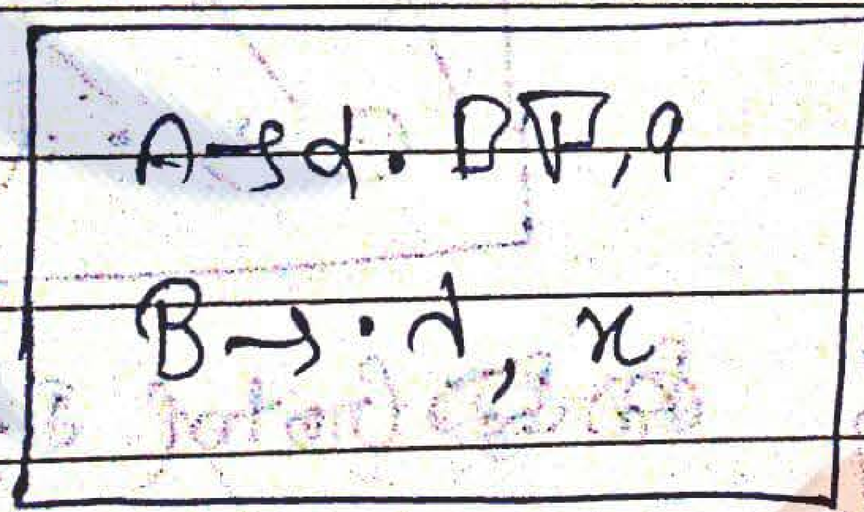
a) construction of closure operations:-

1. Initially add

$S' \rightarrow \cdot S, \$$

2) If $A \rightarrow \alpha \cdot B\beta, a$ is in I_i and if $B = \alpha$ is a production then add $B \rightarrow \cdot\alpha, x$ in I_i where $x \in First(\beta a)$

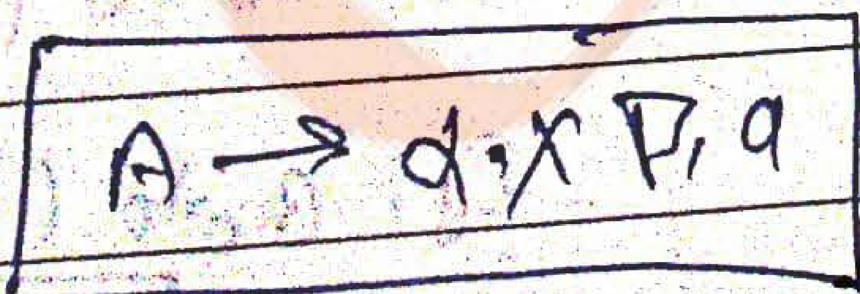
I_i



where $x \in First(\beta a)$

3) If $A \rightarrow \alpha \cdot X\beta, a$ is in I_i, then goto I_i

I_i



then $Goto(I_i, X) = A \rightarrow \alpha X \cdot \beta, a$

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Q) construct LR(1) or CLR(1) parsing table:-

$G = \{S \to CC,\ C \to aC \mid d\}$

I_0

$S' \to \cdot S, \$$
$S \to \cdot CC, \$$
$C \to \cdot aC, a/d$
$C \to \cdot d, a/d$

I_1: Goto (I_0, S)

$S' \to S\cdot, \$$

I_2: Goto (I_0, C)

$S \to C\cdot C, \$$
$C \to \cdot aC, \$$
$C \to \cdot d, \$$

I_3: Goto (I_0, a)

$C \to a\cdot C, a/d$
$C \to \cdot aC, a/d$
$C \to \cdot d, a/d$

I_4: Goto (I_0, d)

$C \to d\cdot, a/d$

I_5: Goto (I_2, C)

$S \to CC\cdot, \$$

~~Goto (I_2, a) = I_3~~

~~$C \to a\cdot C, \$$~~
~~$C \to \cdot aC, \$$~~
~~$C \to \cdot d, \$$~~

I_7: Goto (I_2, d)

$C \to d\cdot, \$$

I_8: Goto (I_3, C)

$C \to aC\cdot, a/d$

~~Goto (I_3, d) = I_4~~

~~$C \to d\cdot, a/d$~~

Goto (I_3, a) = I_3

Goto (I_3, d) = I_4

I_6: Goto (I_2, a)

$C \to a\cdot C, \$$
$C \to \cdot aC, \$$
$C \to \cdot d, \$$

I_9: Goto (I_6, C)

$C \to aC\cdot, \$$

Total states = 10

Goto(I_6, a) = I_6

Goto (I_6, d) = I_7

Q) Construction ... Et

a) ACTION

[I_0] If

[I_1] If

Here

[I_2]

* b)

WWW.GRADESETTER.COM

Scanned with CamScanner

WWW.GRADESETTER.COM

c) Construction of CLR(1) Parsing table

It has two parts:-
a) action
b) Goto

a) ACTION:-

[I.] If I_i $A \to \alpha \cdot a \beta, b$ and

If $Goto(I_i, a) = I_j$

then

Set Action $[i, a] = S_j$

[II.] If

I_i $A \to \alpha \cdot, a$ is in I_i

then

Set action $[i, a]$ = Reduce by $A \to \alpha$

Here, $A \to \alpha$ must not be an augmented production.

[III.] If $S' \to S \cdot, \$$ is in I_i

then

set Action $[i, \$]$ = Accept

* b) Goto:

If I_i $A \to \alpha \cdot B \beta, a$ is in I_i, and

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Page No.
Date

If Goto (I_i, B) = I_j
then
set Goto [i, B] = J

Now give the numbering to each of the Production

1) S → CC
2) C → aC
3) C → d

State	ACTION			GOTO	
	a	d	$	S	C
0	S3	S4		1	2
1			ACCEPT		
2	S6	S7			5
3	S3	S4			8
4	R3	R3			
5			R1		
6	S6	S7			9
7			R3		
8	R2	R2			
9			R2		

In the above table all the entries are single, so
Therefore the grammar is CLR(1)

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Conflicts:-

I.) Shift-Reduce conflict
II.) Reduce-Reduce conflict

I.) Shift-Reduce conflict:-

Ii

$A \to \alpha \cdot a\beta, b$
$B \to \alpha \cdot, a$

⇒ SR conflict
⇒ Not CLR(1)

II.) Reduce-Reduce conflict:-

Ii

$A \to \alpha \cdot, a$
$B \to \alpha \cdot, a$

⇒ R/R conflict
⇒ Not CLR(1)

Q) Check weather the following grammar is CLR(1) or not

1) $G = \{S \to A/a,\ A \to a\}$

$S' \to \cdot S, \$$
$S \to \cdot A, \$$
$S \to \cdot a, \$$
$A \to \cdot a, \$$

same look ahead so Not CLR(1)

$S \to a \cdot, \$$
$A \to a \cdot, \$$

⇒ R/R conflict
⇒ Not CLR(1)

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Page No.
Date

Q) $G : \{ S \to Aa \; , \; A \to Ba / ba \; , \; B \to b \}$

$S' \to \cdot S \; , \$$

$S \to \cdot Aa , \$$

$S \to \cdot Ba , a$

$A \to \cdot ba , a$

$A \to \cdot b , a$

WWW.GRADESETTER.COM

CS Scanned with CamScanner

WWW.GRADESETTER.COM

★ LALR(1) Parsing:-

In the first example of CLR(1), those states having everything is same but different look aheads, we have divides them into two different states but in LALR(1) of parsing, we will merge them into single state.

I_3

$C \to a \cdot C$, a/d
$C \to \cdot aC$, a/d
$C \to \cdot d$, a/d

I_6

$C \to a \cdot C$, \$
$C \to \cdot aC$, \$
$C \to \cdot d$, \$

$I_3 + I_6 = I_{36}$

$C \to a \cdot C$, a/d/\$
$C \to \cdot aC$, a/d/\$
$C \to \cdot d$, a/d/\$

$I_4 + I_7 = I_{47}$

$C \to d \cdot$, a/d/\$

$I_8 + I_9 = I_{89}$

$C \to aC \cdot$, a/d/\$

- Construction of LALR(1) Parsing table is same as Construction of CLR(1) Parsing table

I_0

I_1

I_2

I_{36} ~~I_3 & I_6~~

I_{47} ~~I_4 & I_7~~

I_5 ~~I_8 & I_9~~

I_{89}

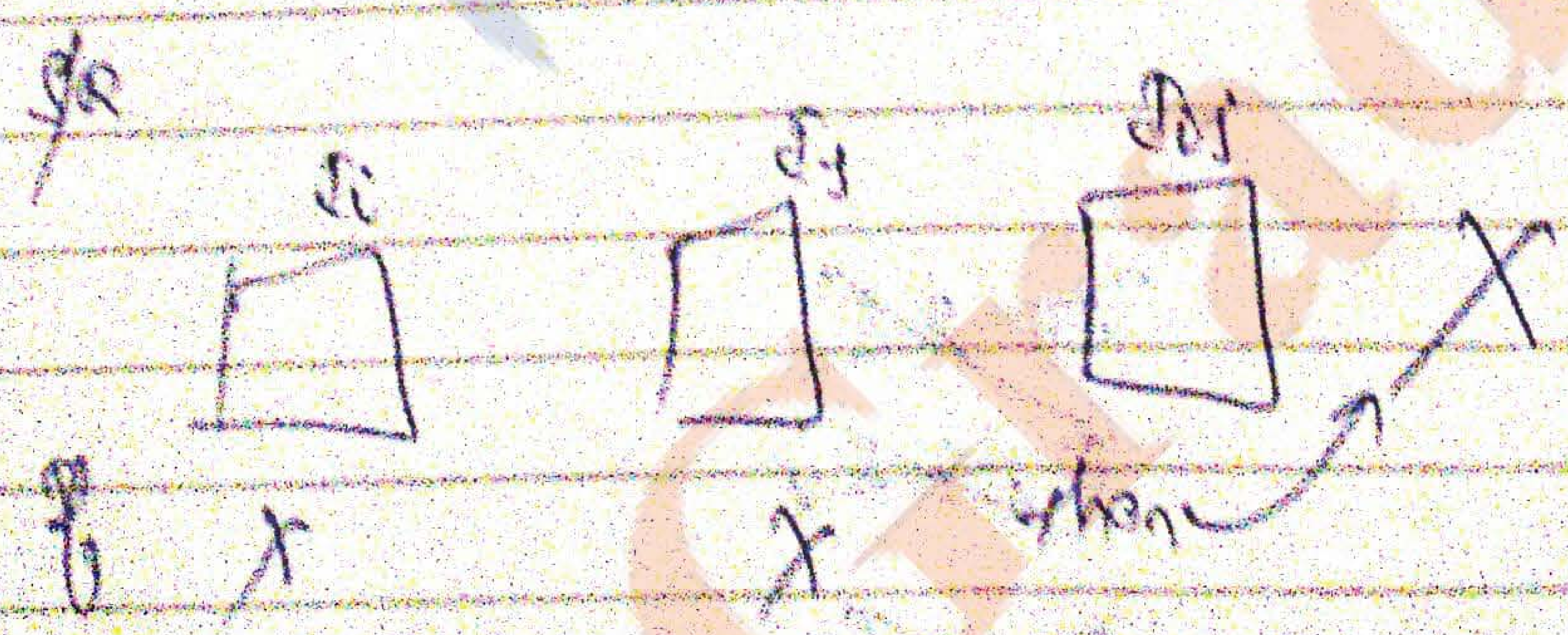
Scanned with CamScanner

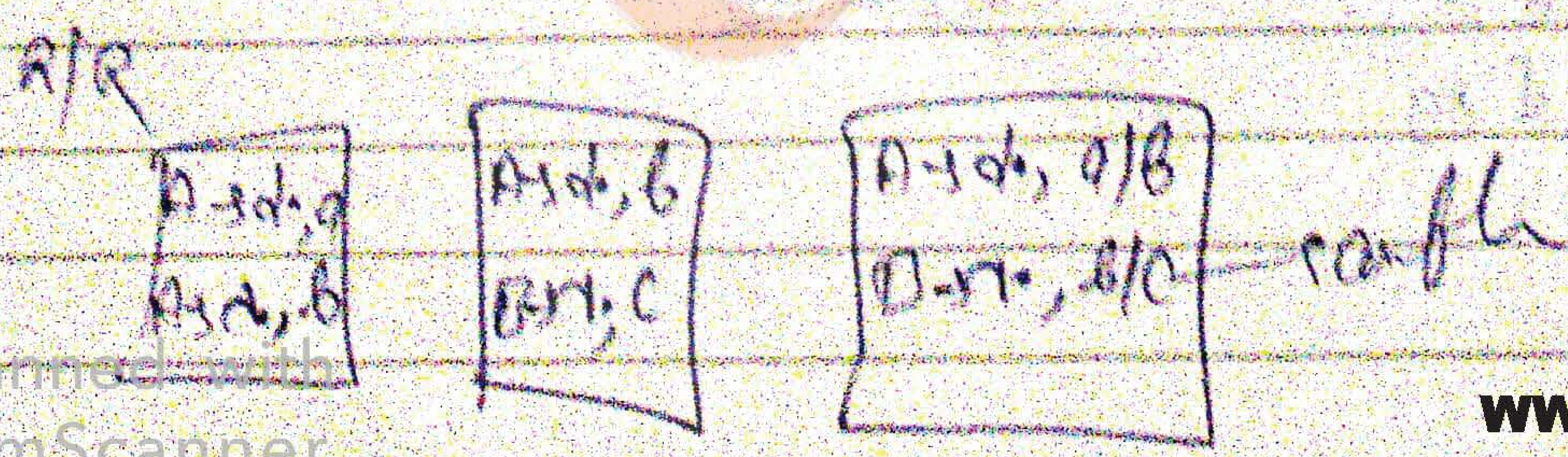
WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

State	ACTION			GOTO	
	a	d	$	S	C
0	S36	S47		1	2
1			ACCEPT		
2	S36	S47			5
36	S36	S47			89
47	R3	R3	R3		
5			R1		
89	R2	R2	R2		

	CLR(1)	LALR(1)
S/R conflict	NO	NO
R/R conflict	NO	Yes/NO





Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Note:

① LR(0) ⊂ SLR(1) ⊂ LALR(1) ⊂ CLR(1)

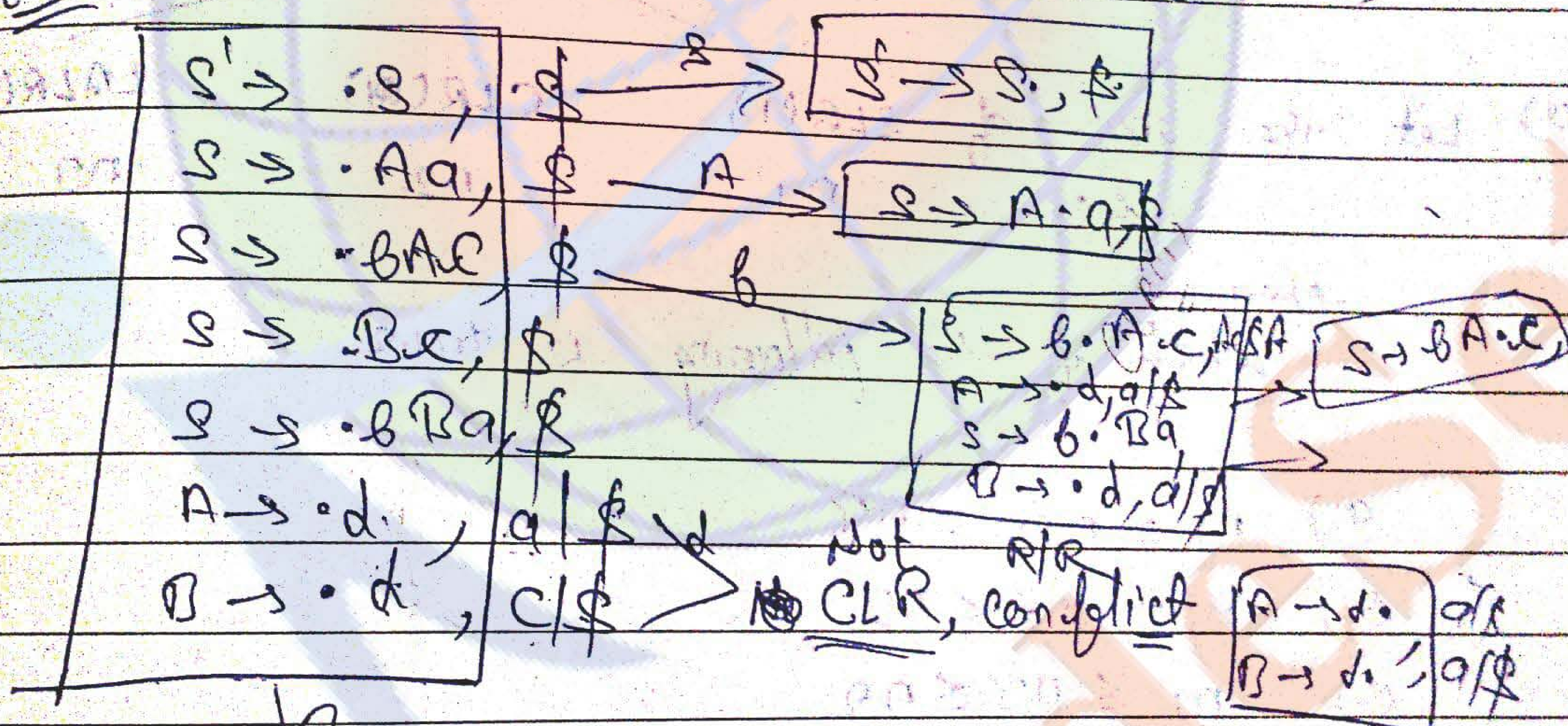
every LR(0) is SLR(1) and every SLR(1) is LALR(1) and every LALR(1) is CLR(1).

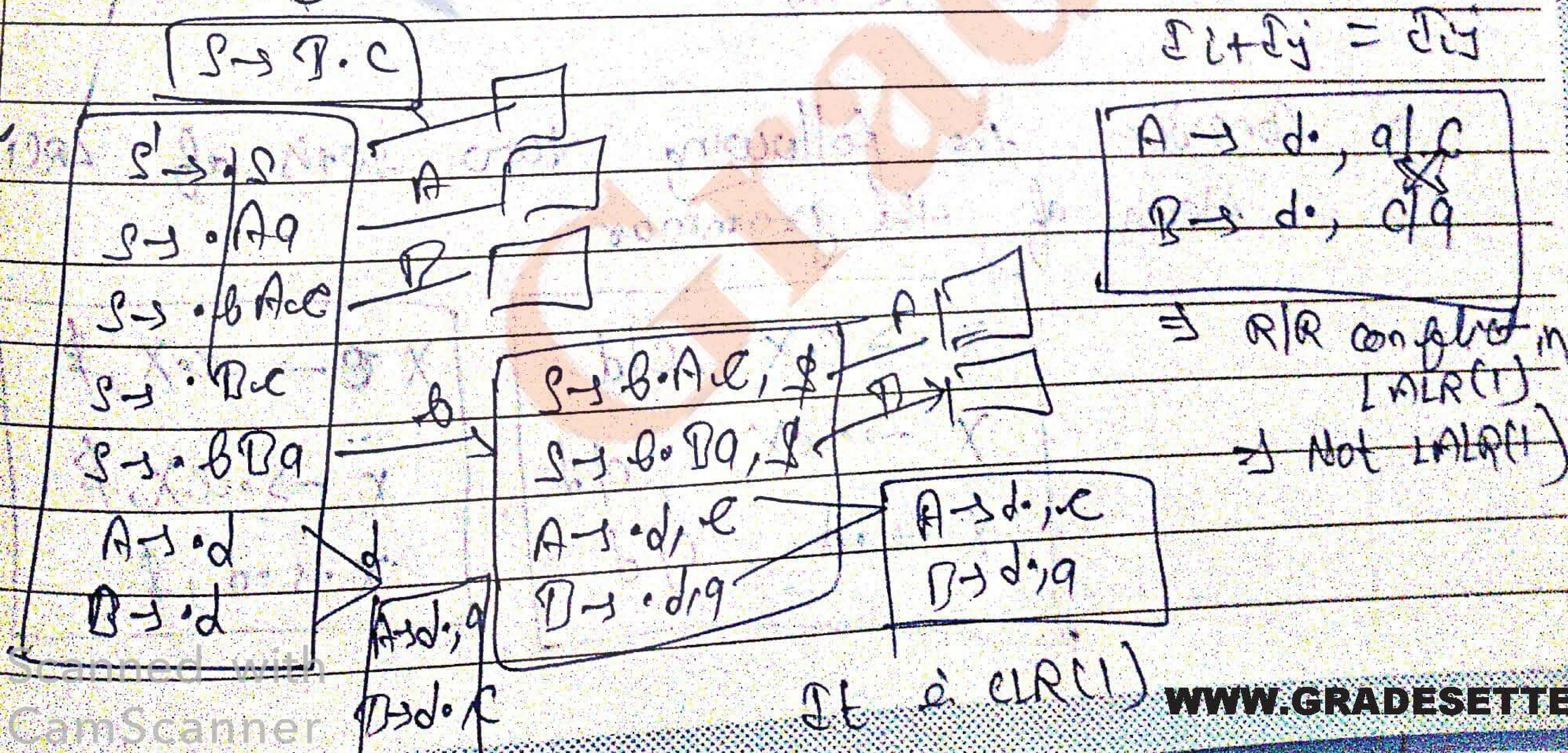
Q) The following grammar is

1) G = { S → Aa / bAc | Bc | bBa , A → d , B → d }

a) LALR(1) but not CLR(1)
b) CLR(1) but not LALR(1)
c) both LALR(1) & CLR(1)
d) Neither LALR(1) nor LR(1)

Sol/ own





$I_i + I_j = I_{ij}$

A → d·, a/c
B → d·, c/a

⇒ R/R conflict in LALR(1)
⇒ Not LALR(1)

It is CLR(1)

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

SLR	LALR(1)	CLR(1)
1) SLR is the smaller in size	1) The size of LALR & same as SLR	1) CLR is largest size
2) SLR is an easy method based on follow function	2) The method can be applicable more wider class then SLR	2) CLR is the most powerful method then SLR & LALR
3) Error detection is not immediate	3) Not immediate	3) Immediate error detection

Q) Let the sizes of SLR(1), CLR(1), LALR(1) be n_1, n_2, n_3 then which of the following is true

a) $n_1 = n_2 = n_3$
b) $n_1 = n_2 < n_3$
c) $n_1 \leq n_3 \leq n_2$
d) $n_1 = n_3 < n_2$

Note

Q) Consider the following two sets of LR(1) items of the grammar

$X \rightarrow c.X \;, c/d$
$X \rightarrow .cX \;, c/d$
$X \rightarrow .d \;, c/d$

$X \rightarrow c.X \;, \$$
$X \rightarrow .cX \;, \$$
$X \rightarrow .d \;, \$$

regarding ... in LA... whic...

a) ...
b) ...
c) ...
d) ...

Soln

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Page No.
Date

regarding the merging in LALR(1) of the above two set's of parsing which of the following is/are False.

a) can't mergable since look a heads different
b) can be mergable but will result in S/R conflict
c) " " R/R conflict
d) can't be mergable since goto on C will leads to two different states.

a) b) c) d) 1, 2, 3, 4

a) ✓
b) ✗
c)
d) ✗

Goto(C)

$X \to C \cdot X$, c|d|\$
$X \to \cdot CX$, c|d|\$
$X \to \cdot d$, c|d|\$

यहाँ पर सब First C
जो final है ही नहीं
तो S/R, or R/R की
बात नहीं है।

Note:

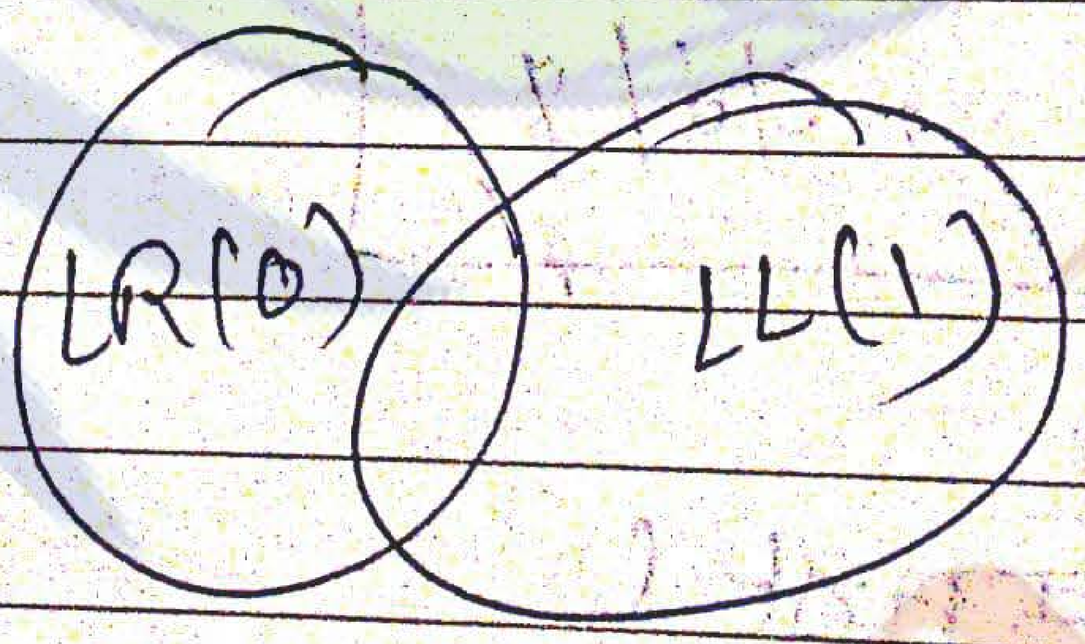
CLR(1)
LALR(1)
SLR(1)
LR(0)
LL(1)

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

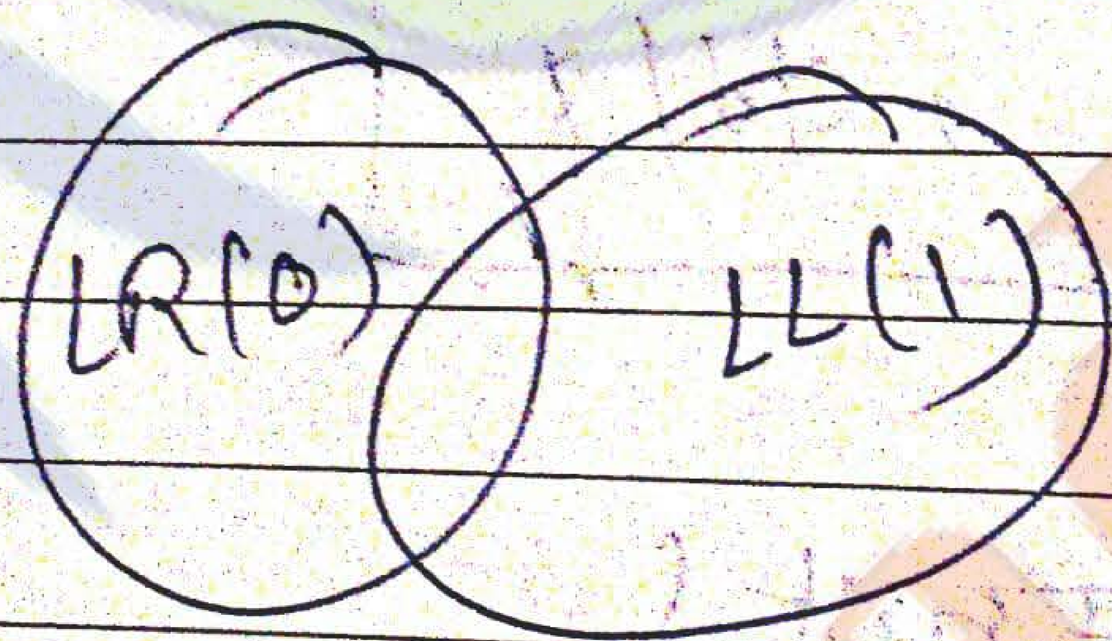
कहीं पर कुछ मिला हुआ भी होता है



Scanned with CamScanner
WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

कहीं पर कुछ तीन दूसरा भी होता है



WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Semantic Analysis:

> Note:
> Grammar + Semantic rule = Syntax directed translation (SDT)

The tokens generated by Lexical Analyser will be grouped together in Syntax analysis to form a parse tree, it can not check the meaning of the source code. The meaning of the source code will be verified in semantic analysis.

This can be done attaching some rules to every production of the grammar.

The grammer with semantic rules is called Syntax

\# consider the following grammar with semantic rules

~~i/p = 1+2*3~~

E → E+E { E.val = E.val + E.val }

E → E*E { E.val = E.val * E.val }

E → id { E.val = id.val }

if the input is
i/p = 1+2*3 then o/p is

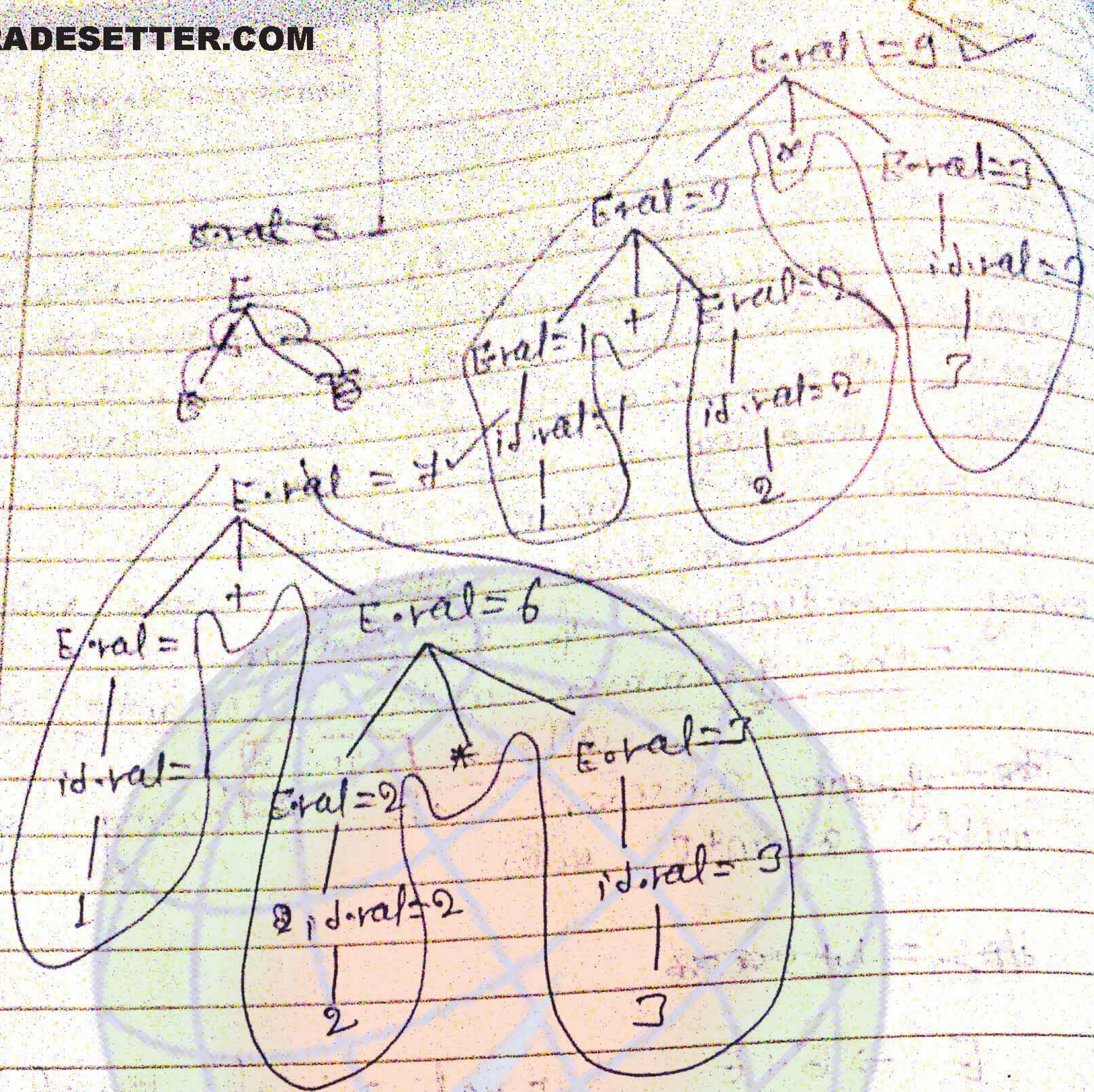
a) 7 b) 9 c) both 7 & 9 ✓, (d) none
↓
because ambiguous

1+2*3
id+id*id

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM



Ambiguous

Q2.) consider the following grammar with semantic rule.

E → E + T { E.val = E₁.val + T.val }

E → T { E.val = T.val }

T → T * F { T.val = T₁.val * F.val }

T → F { T.val = F.val }

F → id { F.val = id.val }

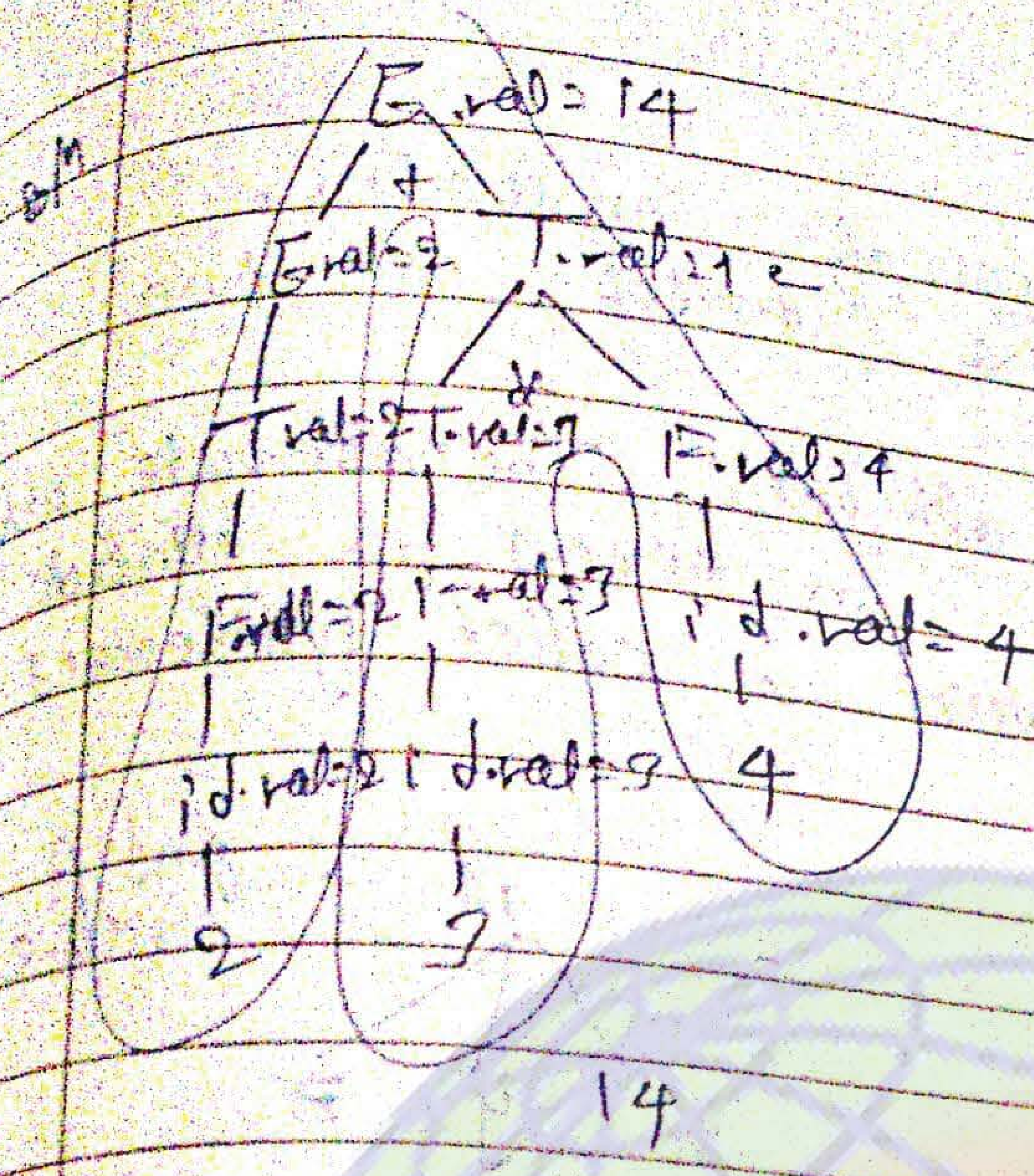
If i/p = 2+3*4 then o/p is

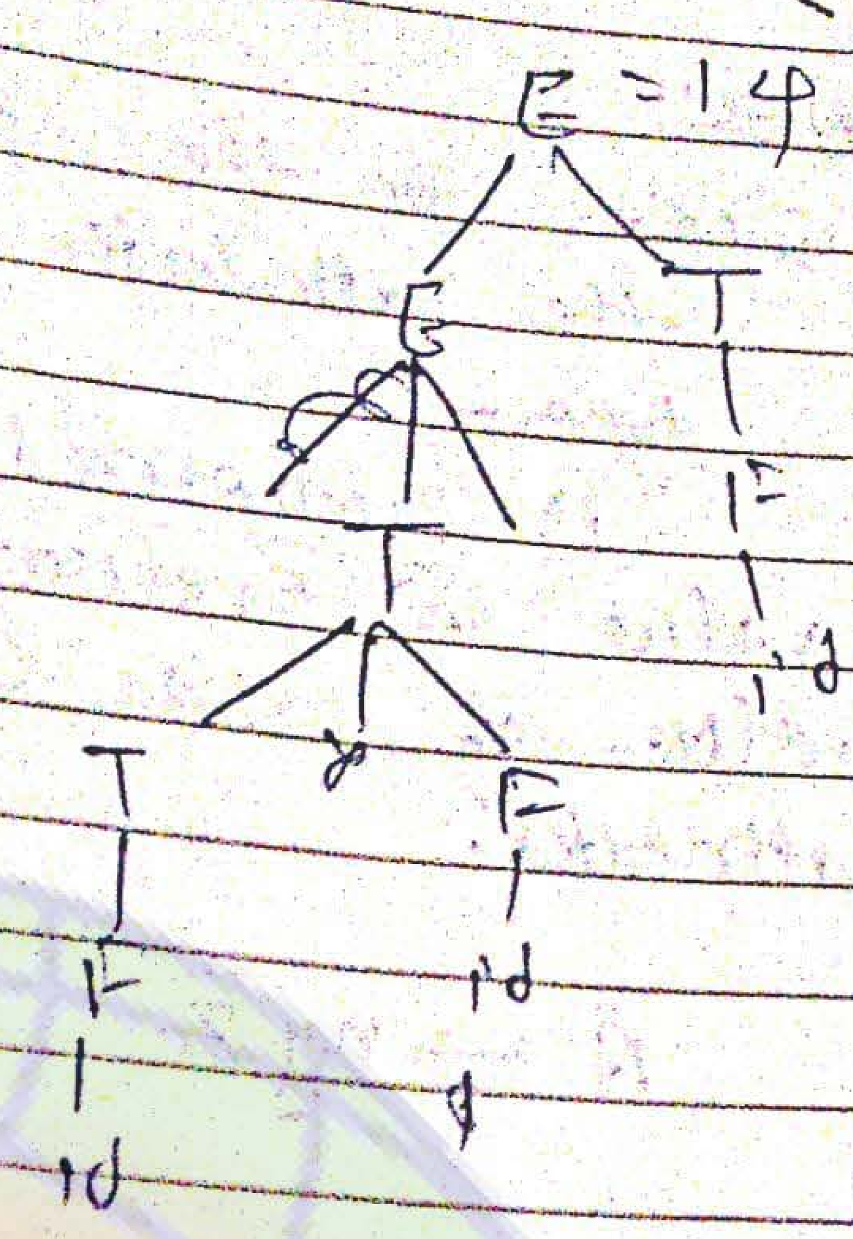
a) 14 b) 20 c) 14 & 20 d) None

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM





Q) Consider the following grammar with semantic rule.

E → E+T { printf ("+"); }

E → T { }

T → T*F { printf ("*"); }

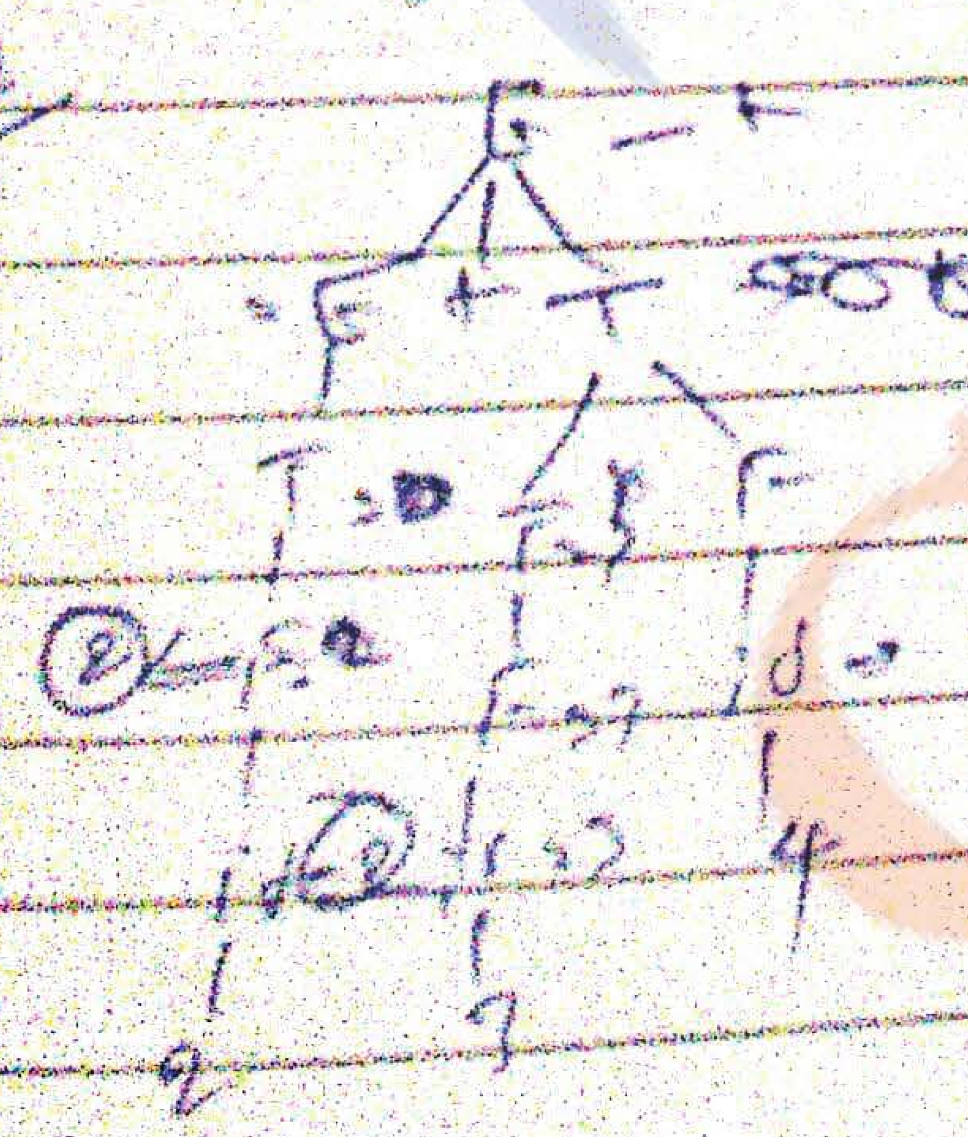
T → F { }

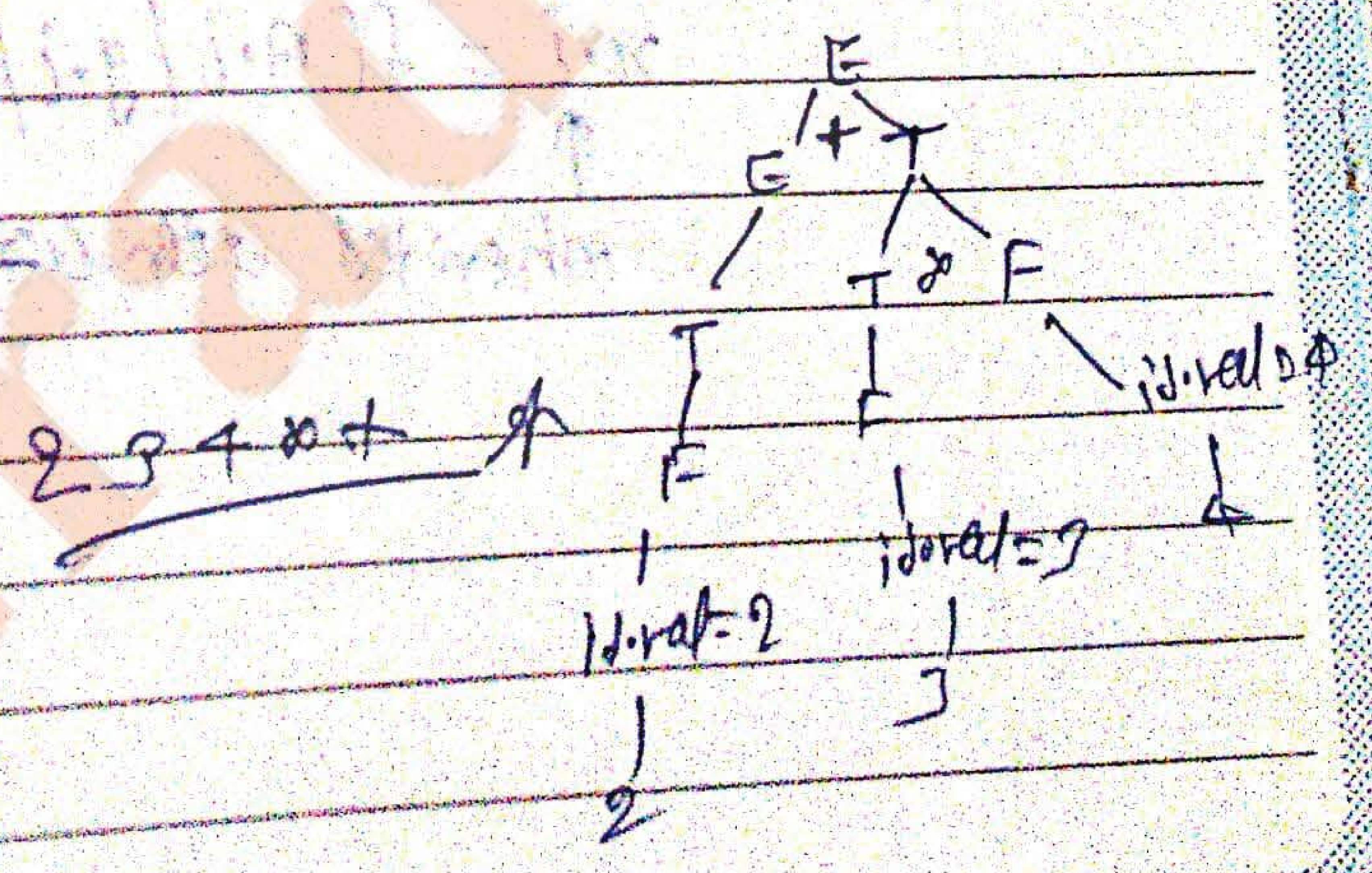
F → id { printf(id.val); }

If i/p = 2+3*4 then o/p = ?

a) 234+* b) 234*+ c) 23*4+ d) 14

Ans

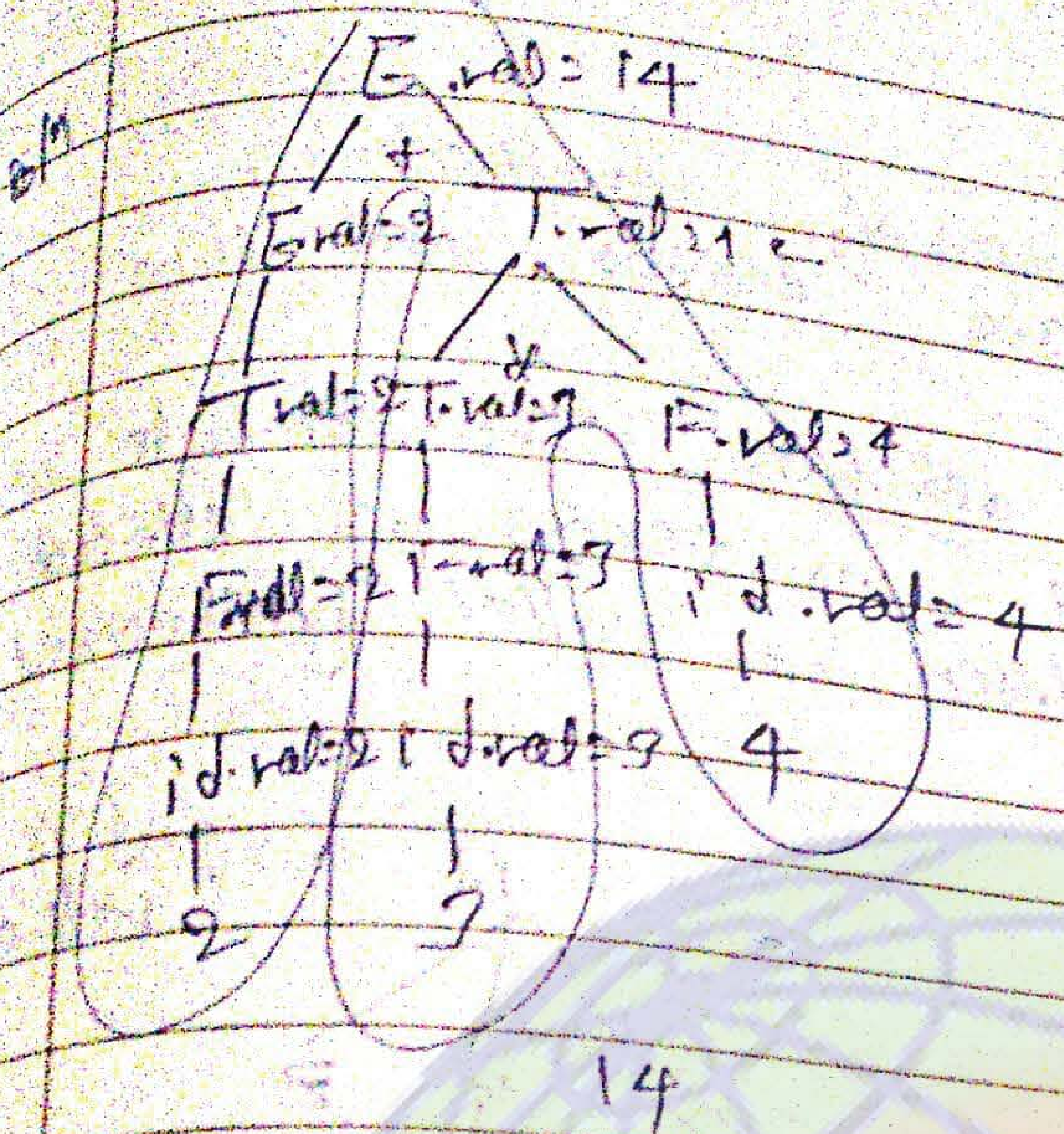


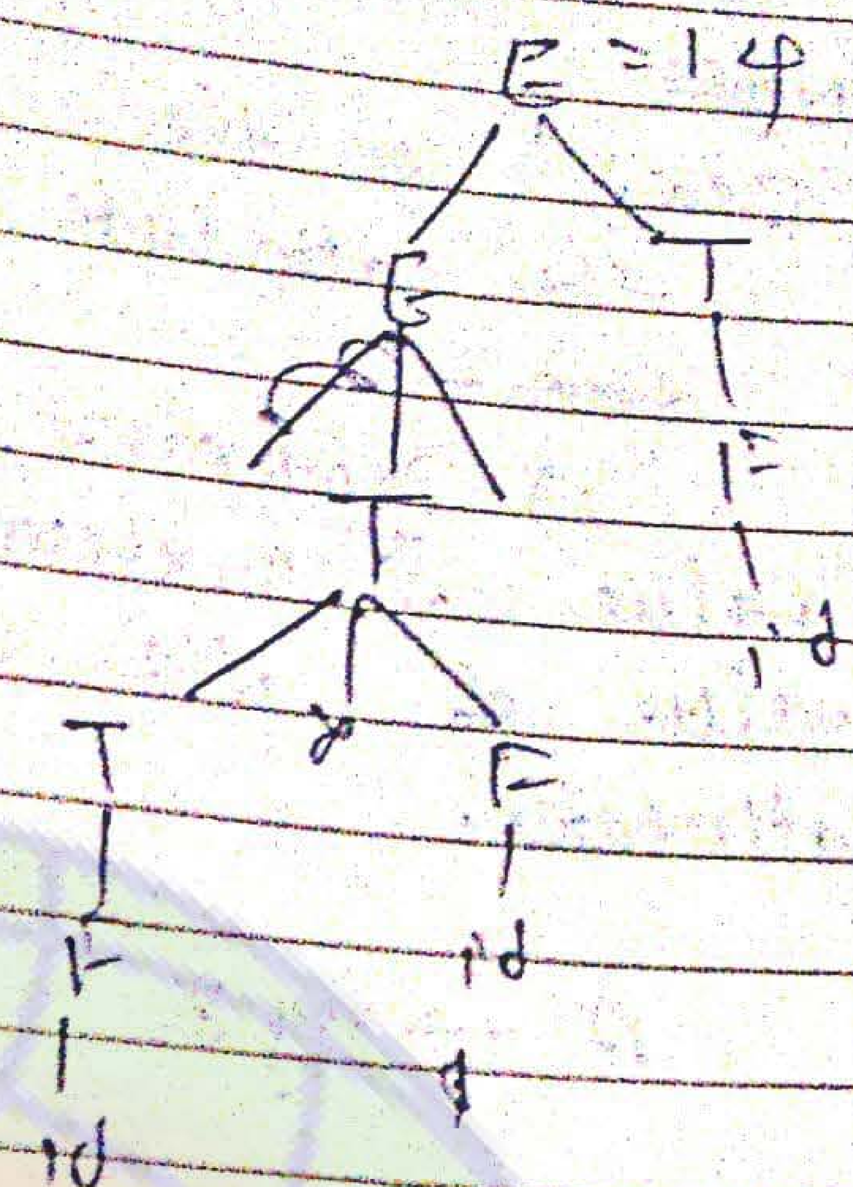


Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM





14

14

Q) Consider the following grammar with semantic rule.

E → E+T { printf ("+"); }

E → T { }

T → T*F { printf ("*"); }

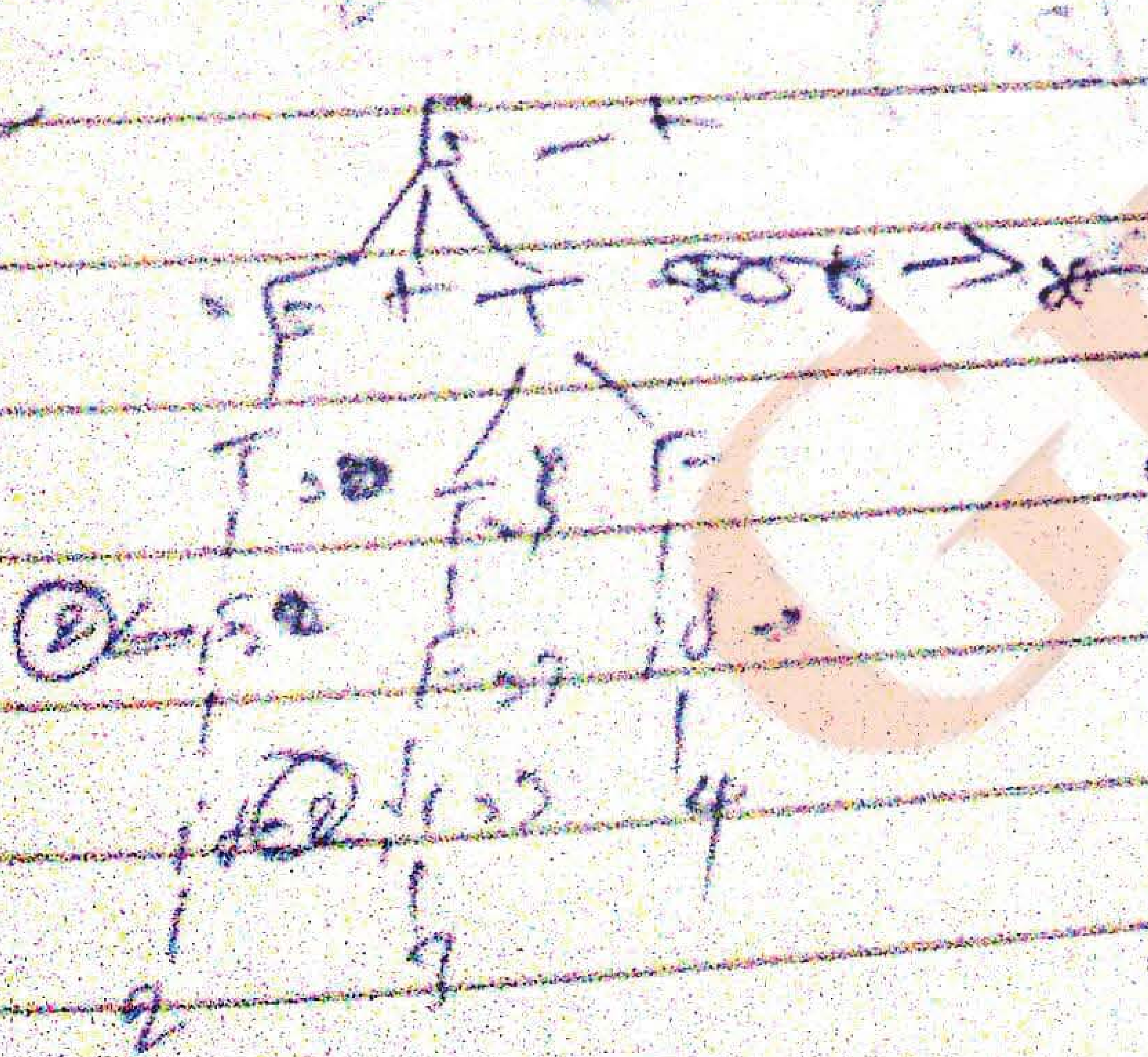
T → F { }

F → id { printf(id.val); }

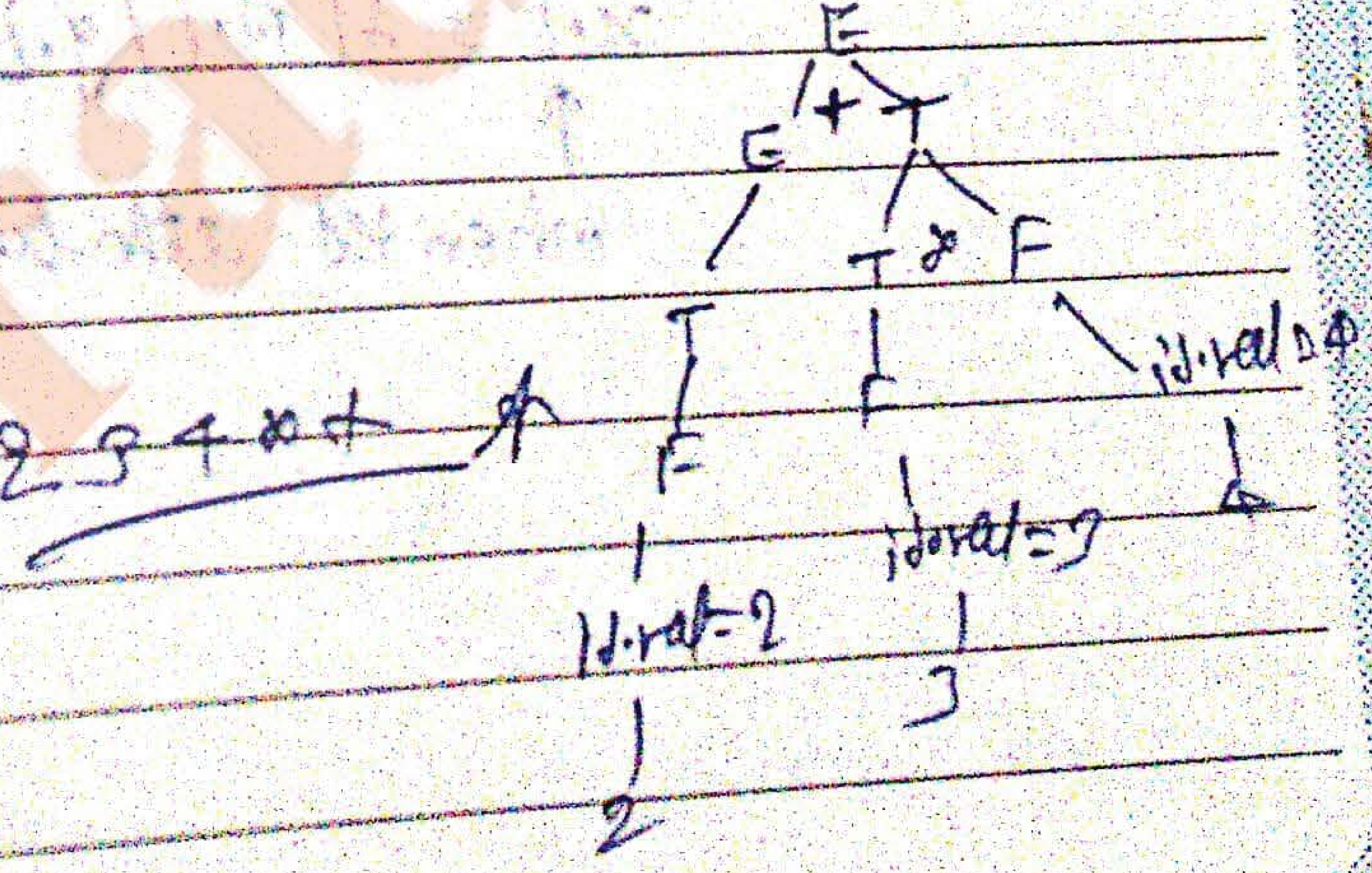
If i/p = 2+3*4 then o/p = ?

a) 234+* b) 234*+ c) 23*4+ d) 14

Ans



234*+



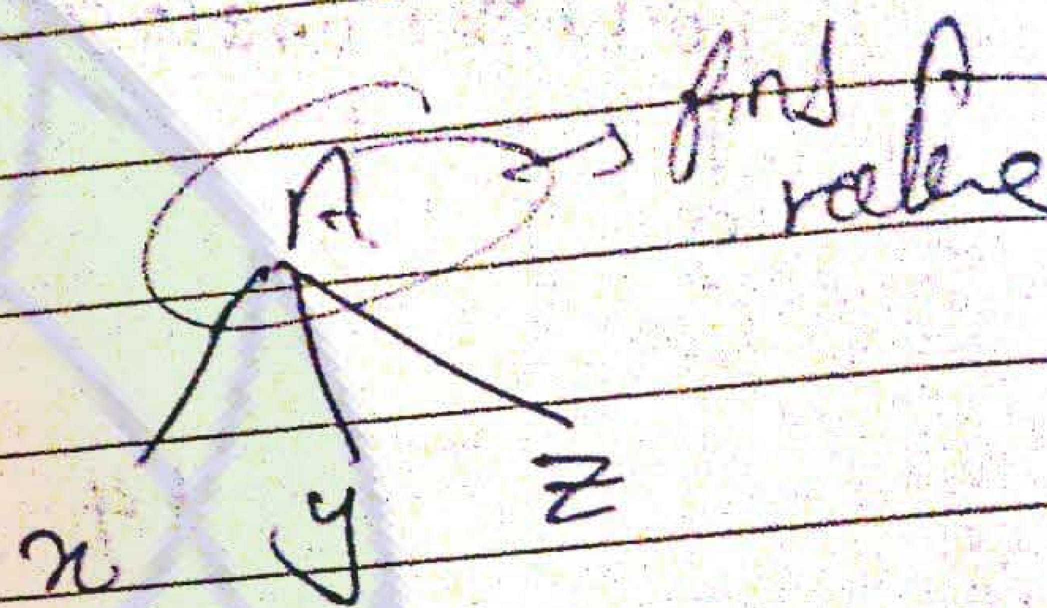
WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Note:
Based on the process of evaluation, the attributes are divided into two types –

1) Synthesized attribute: The attribute whose value is evaluated in terms of attribute values of its children is called synthesized attributes.

$A \rightarrow xyz$

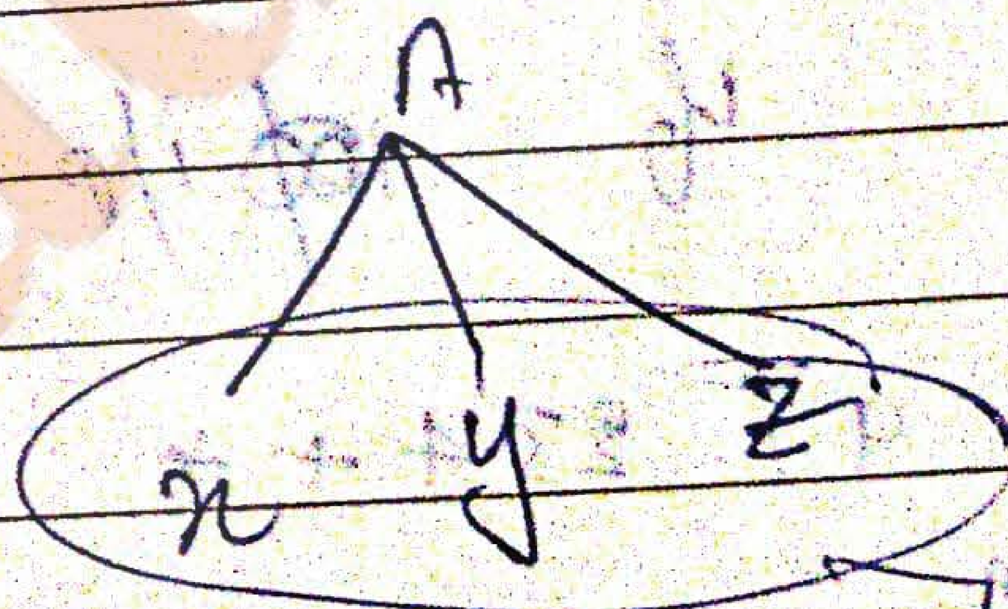


$A.s = f(x.s \mid y.s \mid z.s)$

↑ synthesized attributes.

2) Inherited attributes: –
The attribute whose value is evaluated in terms of its parent or siblings is called inherited attribute

$A \rightarrow xyz$



$x.i = f(A.i \mid y.i \mid z.i)$

↑ inherited attribute.

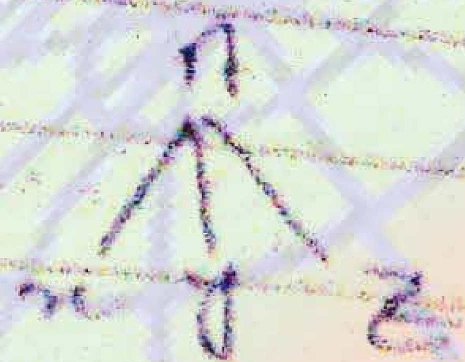
Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

(Every S-attribute is L-attribute) R1

- S-attribute:
  S-attribute is synthesized attribute.
- L-attribute:
  L-attribute is both synthesized and inherited attribute.
  If it is inherited attribute, it must inherit the values from the it's parent or left side of the siblings only. R1



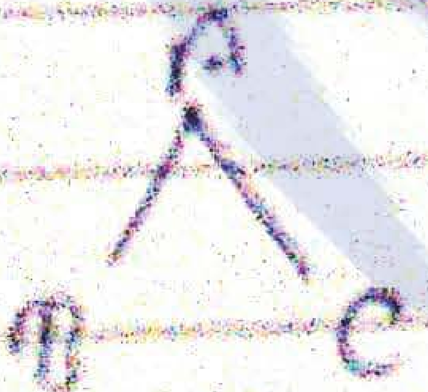
$y.i = (A.i)$ ✓

$x.i = (y.i)$ ✗ (Not L-attr)

Q) The following grammar is

a) $A \rightarrow BC$ { $A.s = f(B.s)$ & $C.i = f(A.i)$ }

a) S-attr b) L-attr c) both a & b d) None



$A.s = f(B.s)$ & $C.i = f(A.i)$

L-attr ✓ Not S ✗

S-attr ✓ L-attr ✓ ✓

Scanned with CamScanner

WWW.GRADESETTER.COM

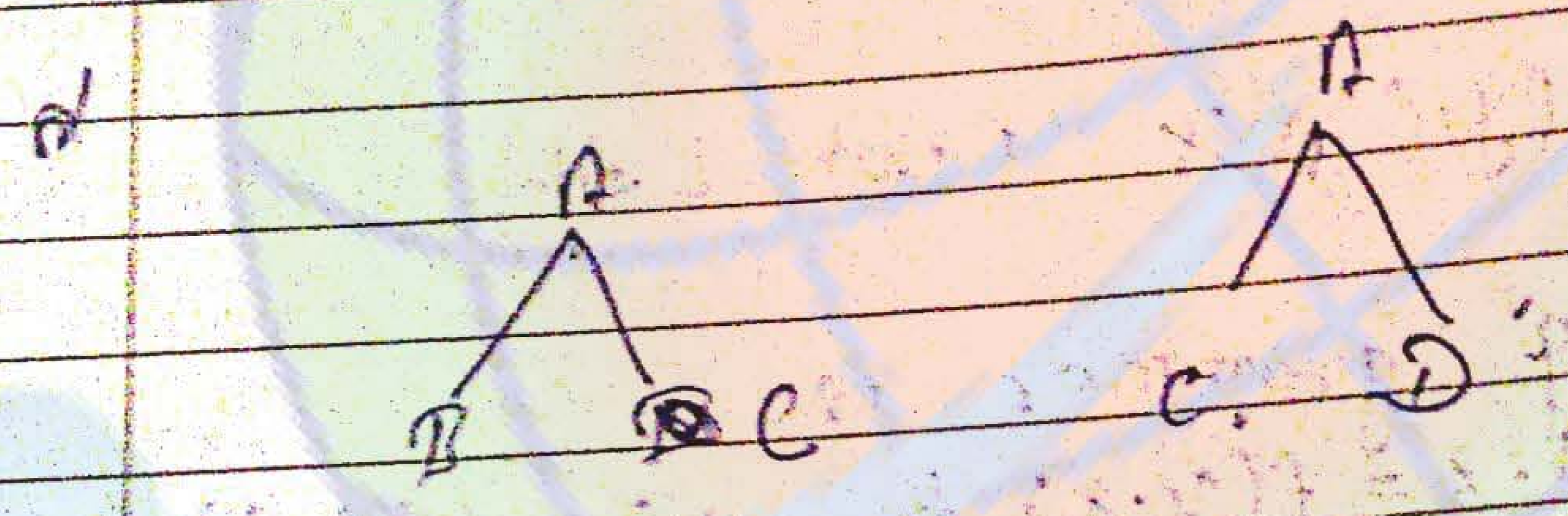
WWW.GRADESETTER.COM

Page No. ___
Date ___

c) A → BC { B.i = f(A.i); — Not S
C.i = f(B.i); — L
A.s = f(B.s) } — L

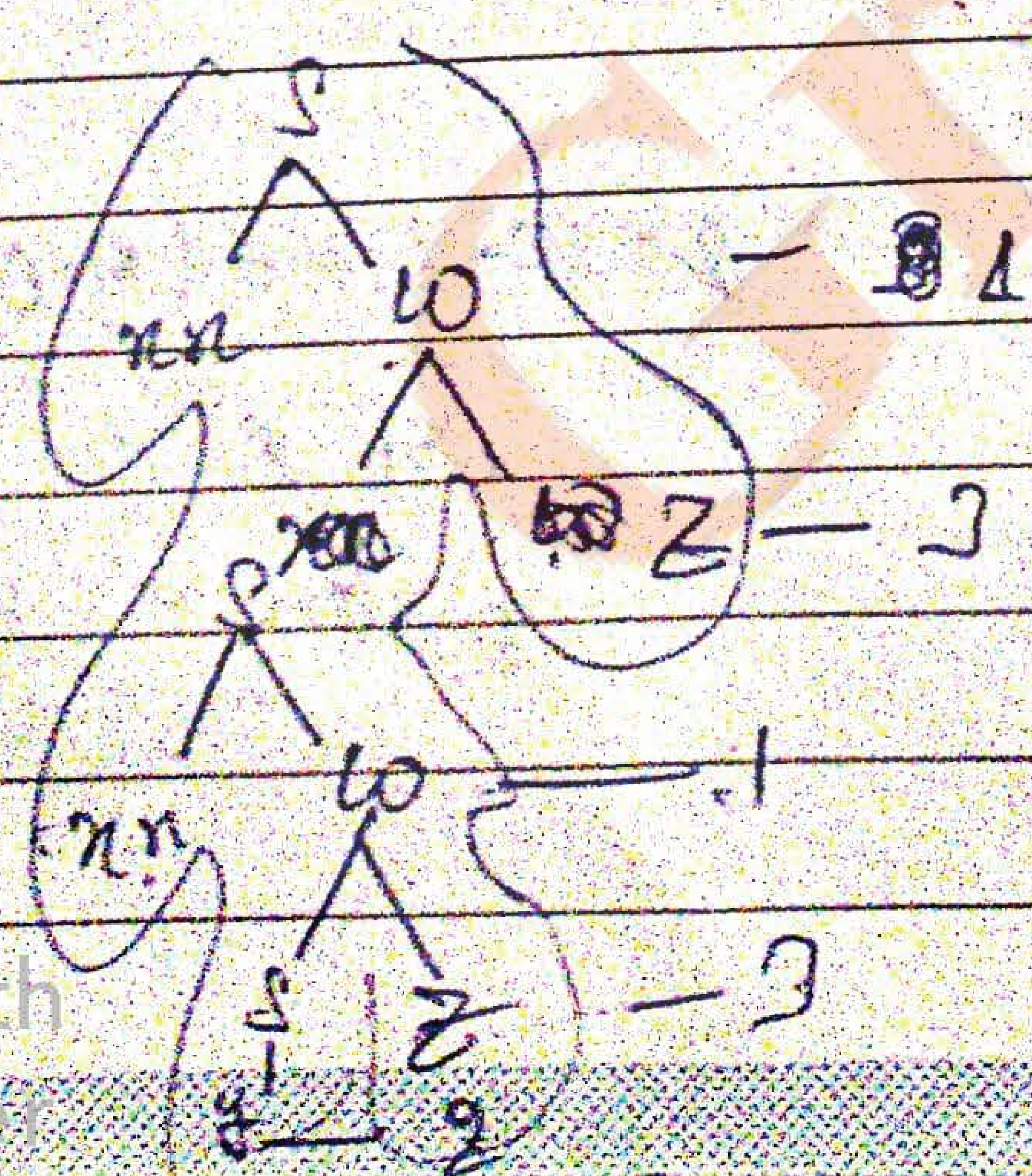
A → CD { A.s → f(D.s); — Not L
C.i = f(D.i);
D.i = f(A.i); }

a) S-attr b) L-attr c) both a & b d) None



Q.) S → xxW { printf(1); }
| y { Print(2); }
W → Sz { Print(3); }

i/p = xxxxyzz
then o/p = ?



— 1
— 3
23131

Consider ...
a) ↑ less powerful ... more powerful
E → E*T
| T
T → F ...
| F
F → ...
| ...
if
Ans:

Scanned with CamScanner
WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Page No.
Date

Q) consider the following

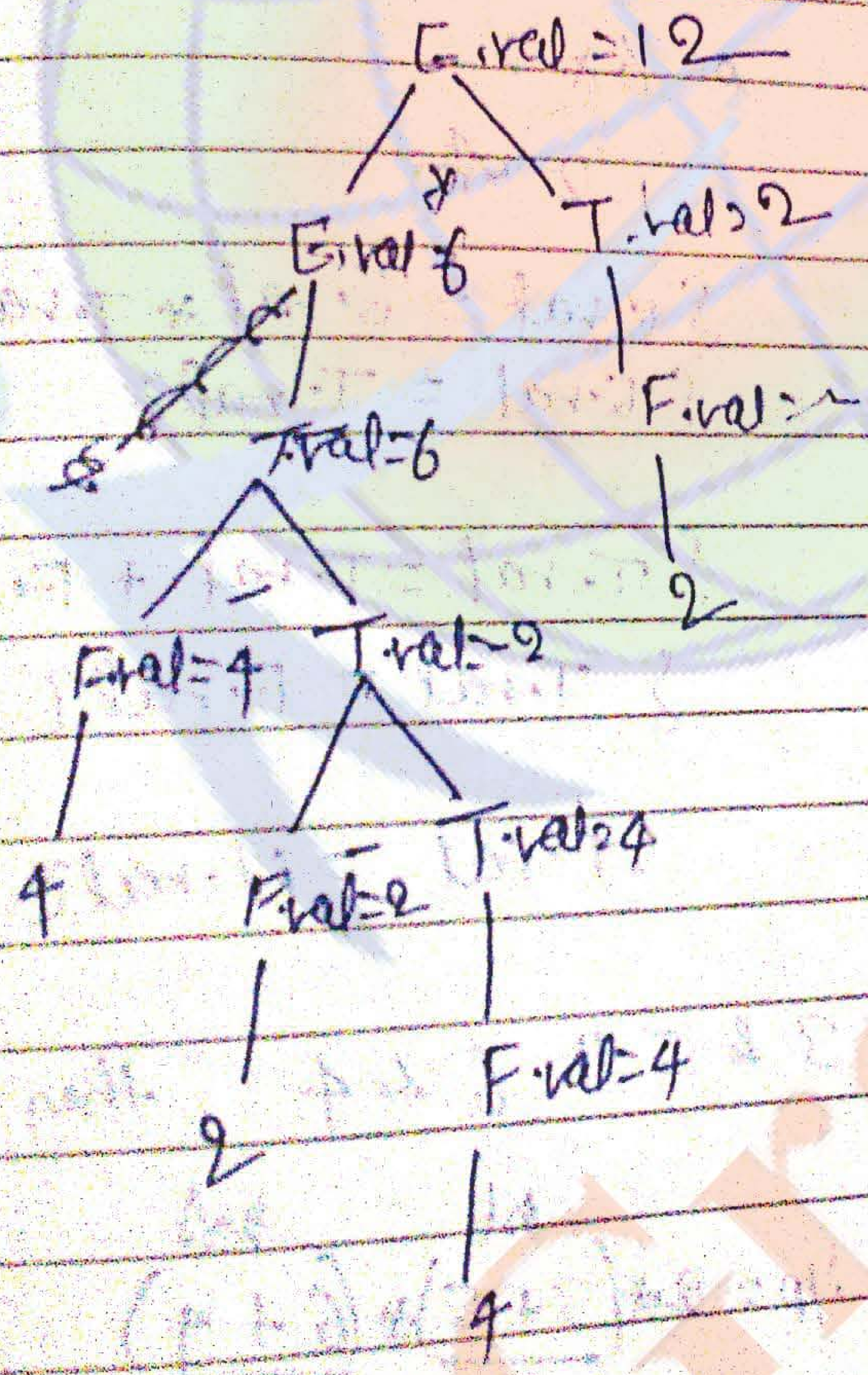
E → E*T { E.val = E.val * T.val }
 | T { E.val = T.val }

T → F − T { T.val = F.val − T.val }
 | F { T.val = F.val }

F → 2 { F.val = 2 }
 | 4 { F.val = 4 }

if i/p = 4 − 2 − 4 * 2 then o/p = ?

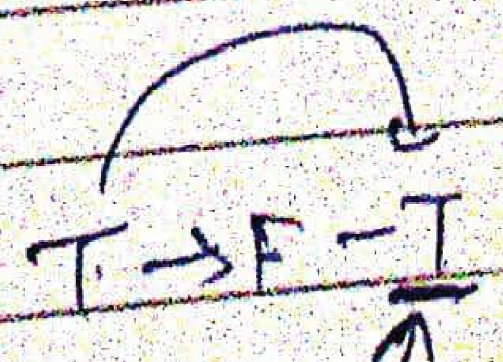
Ans:



short cut:
The lower operator is highest priority

− > *

− is right associative



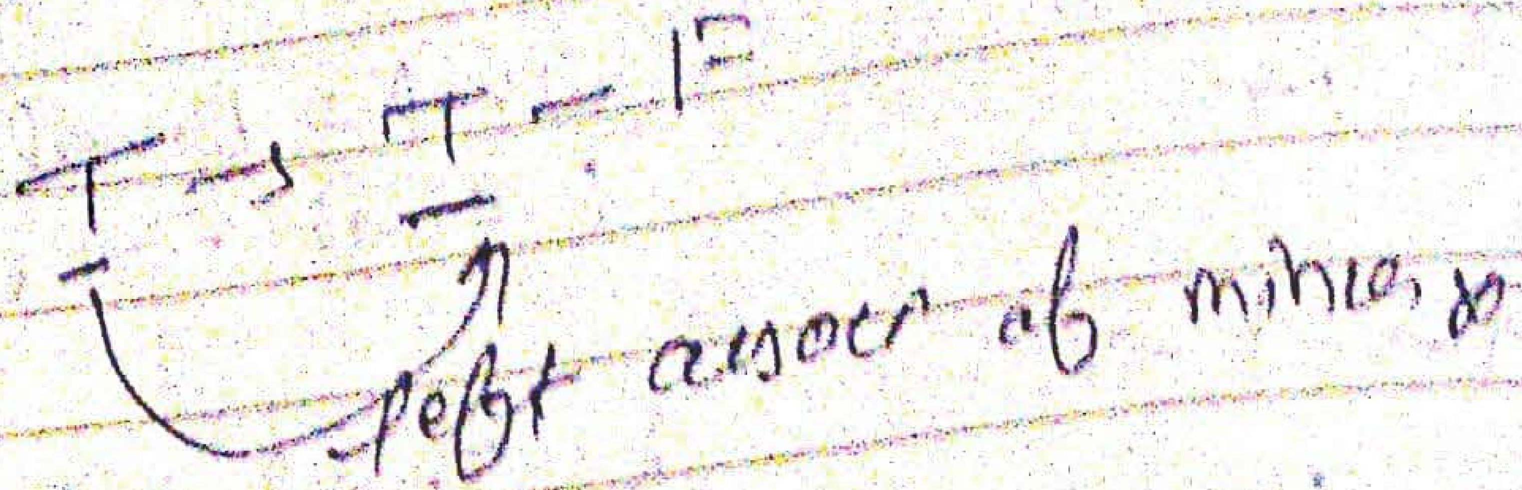
WWW.GRADESETTER.COM

Scanned with CamScanner

WWW.GRADESETTER.COM

Page No.
Date

Note.



1st

$((4-(2-4))*2)$

$(4-(-2))*2$

$= 6*2$

12 Ans

Q.) E → E # T { E.val = E.val * T.val }
 | T { E.val = T.val }

T → T & F { T.val = T.val + F.val }
 | F { T.val = F.val }

F → id { F.val = id.val }

i/p = 2 # 3 & 5 # 6 & 4 then o/p = ?

soln

& + (1st) left i/p = 2 * (3 + 5) * (6 + 4)
 1st 2nd
 8 10

* (left)

2 * 8 * 10

16 * 10

160 Ans

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Page No
Date

c) N → L { N.val = L.val }

L → L B { L.val = L.val + B.val }

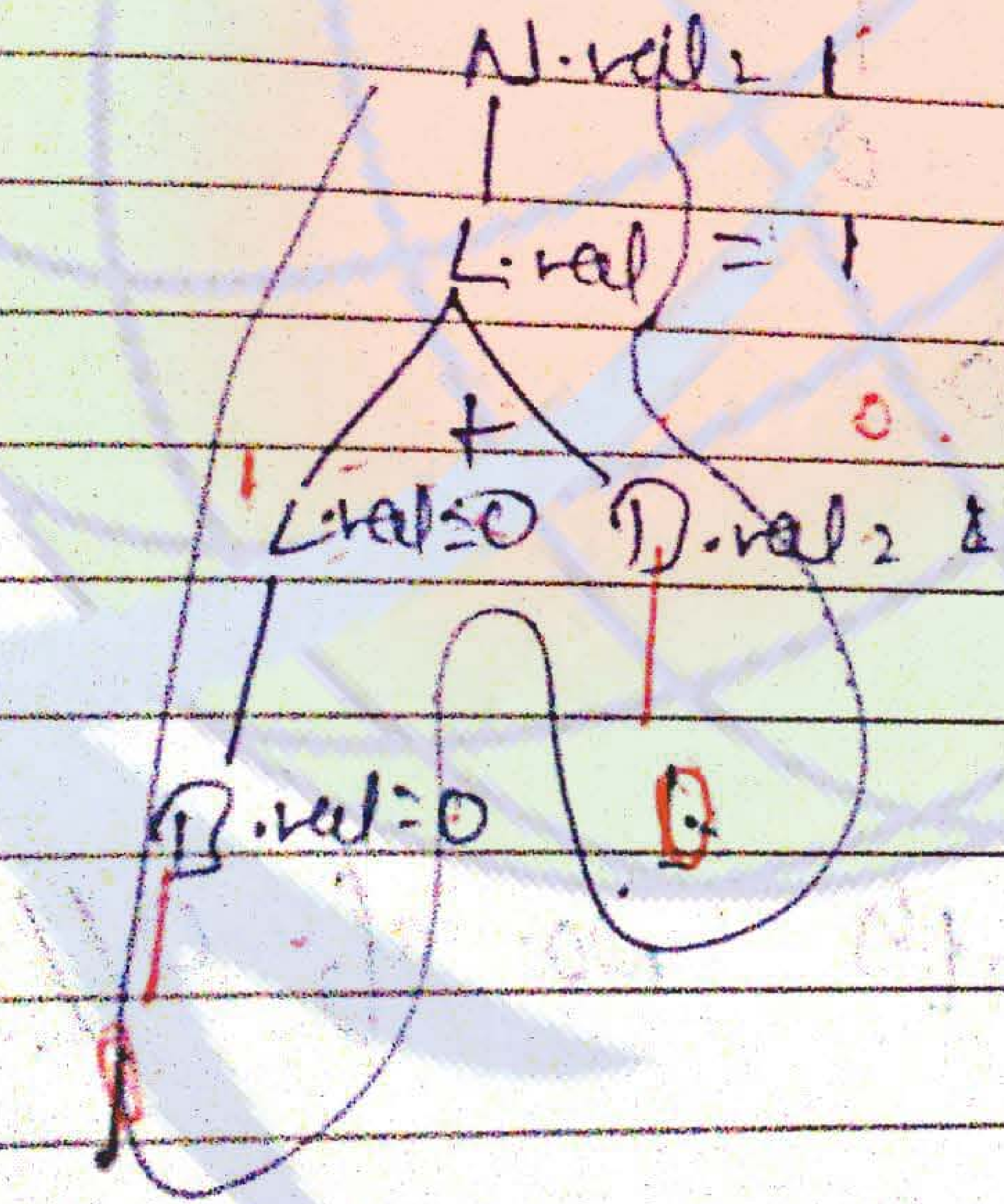
| B { L.val = B.val }

B → 0 { B.val = 1 }

| 1 { B.val = 0 }

what is the purpose of the grammar.

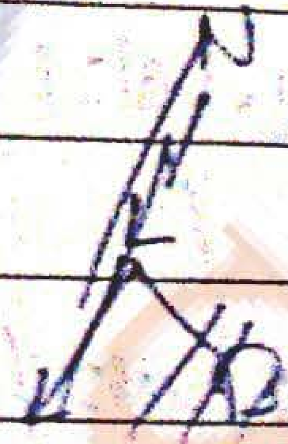
10/10

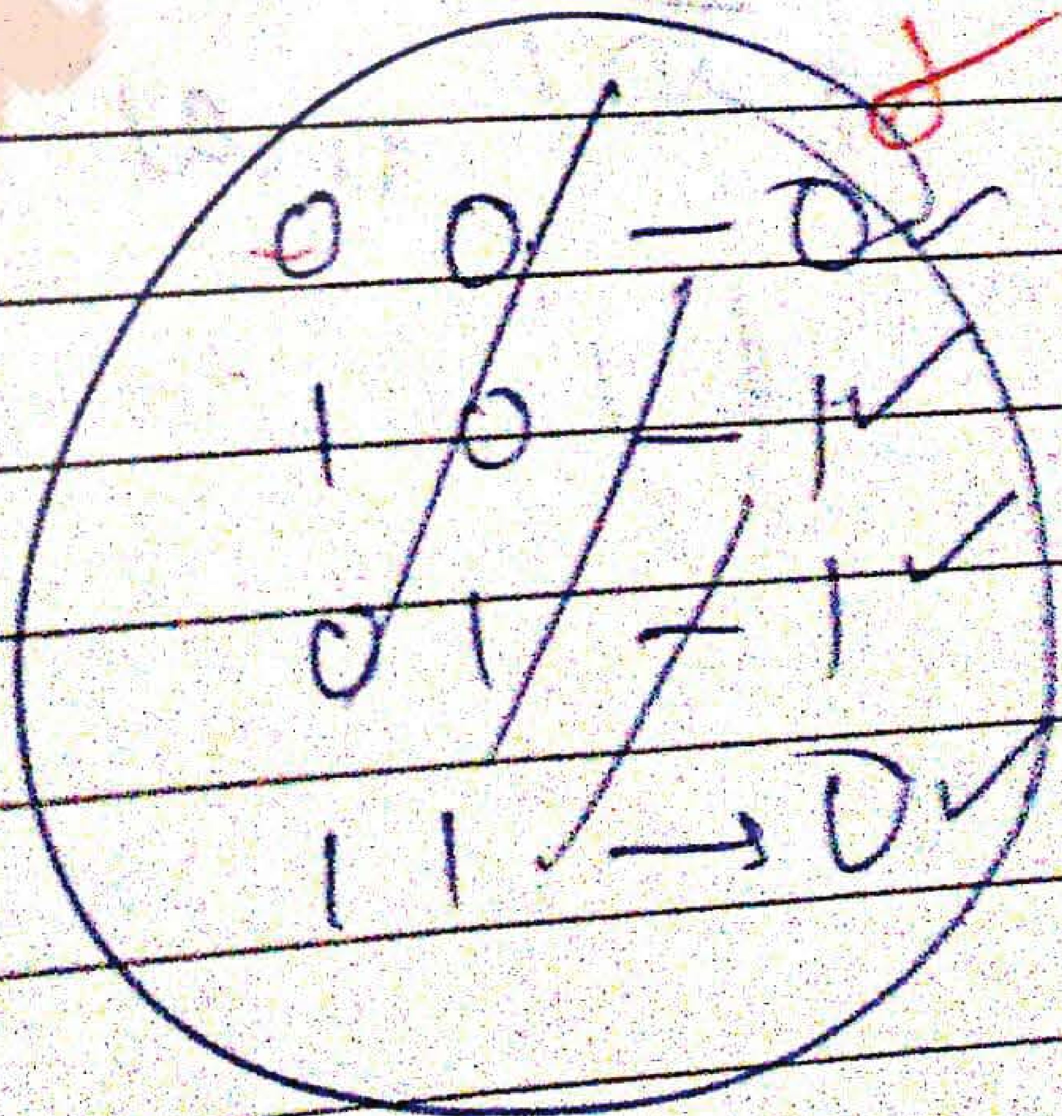


take any input

0 1 0 0

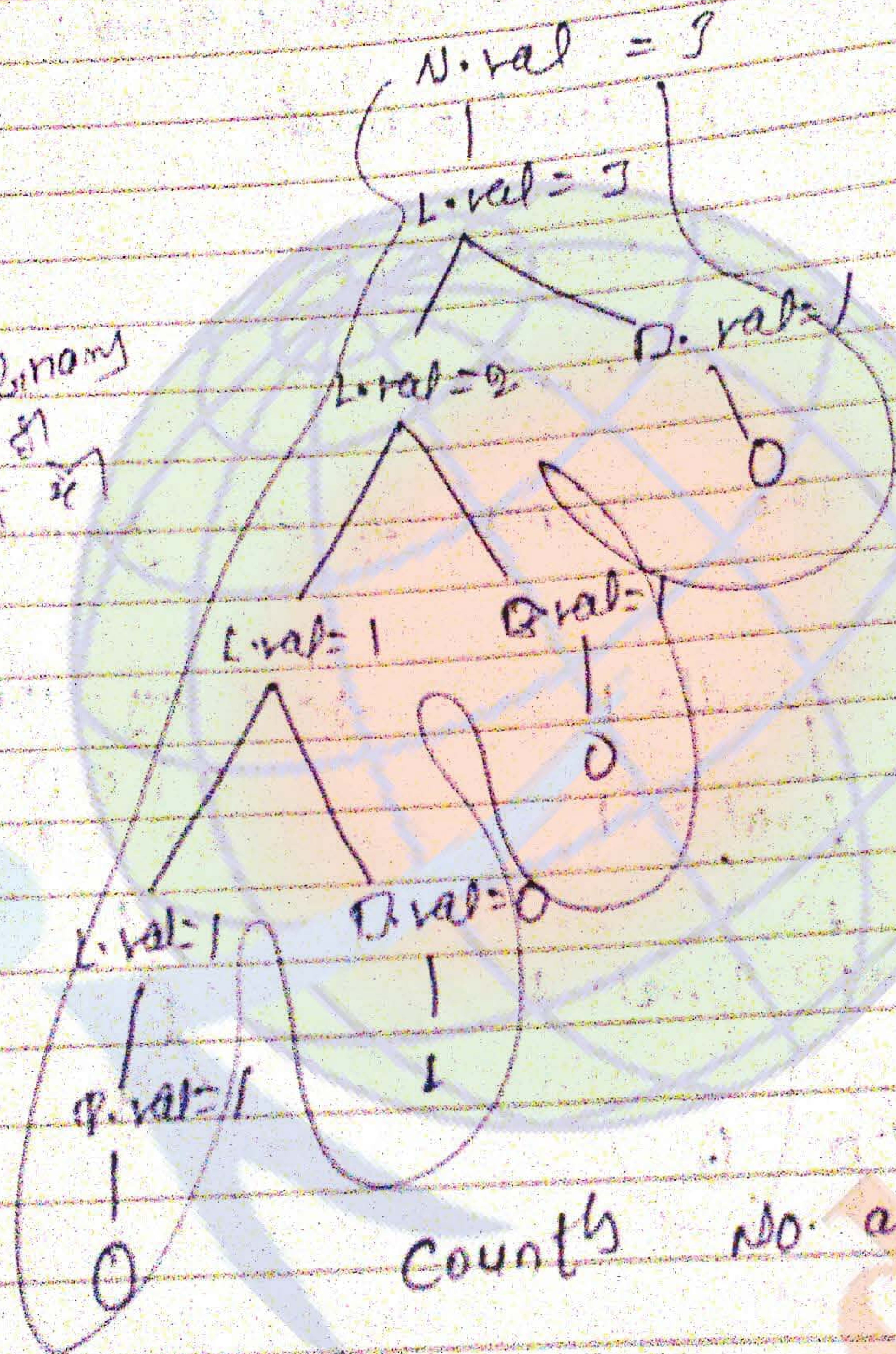
B B B B





Scanned with CamScanner

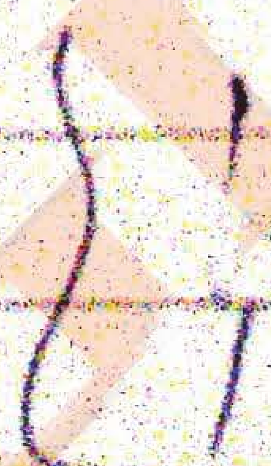
WWW.GRADESETTER.COM



Counts no. of 0//

Note:

D-SD/ R.val —

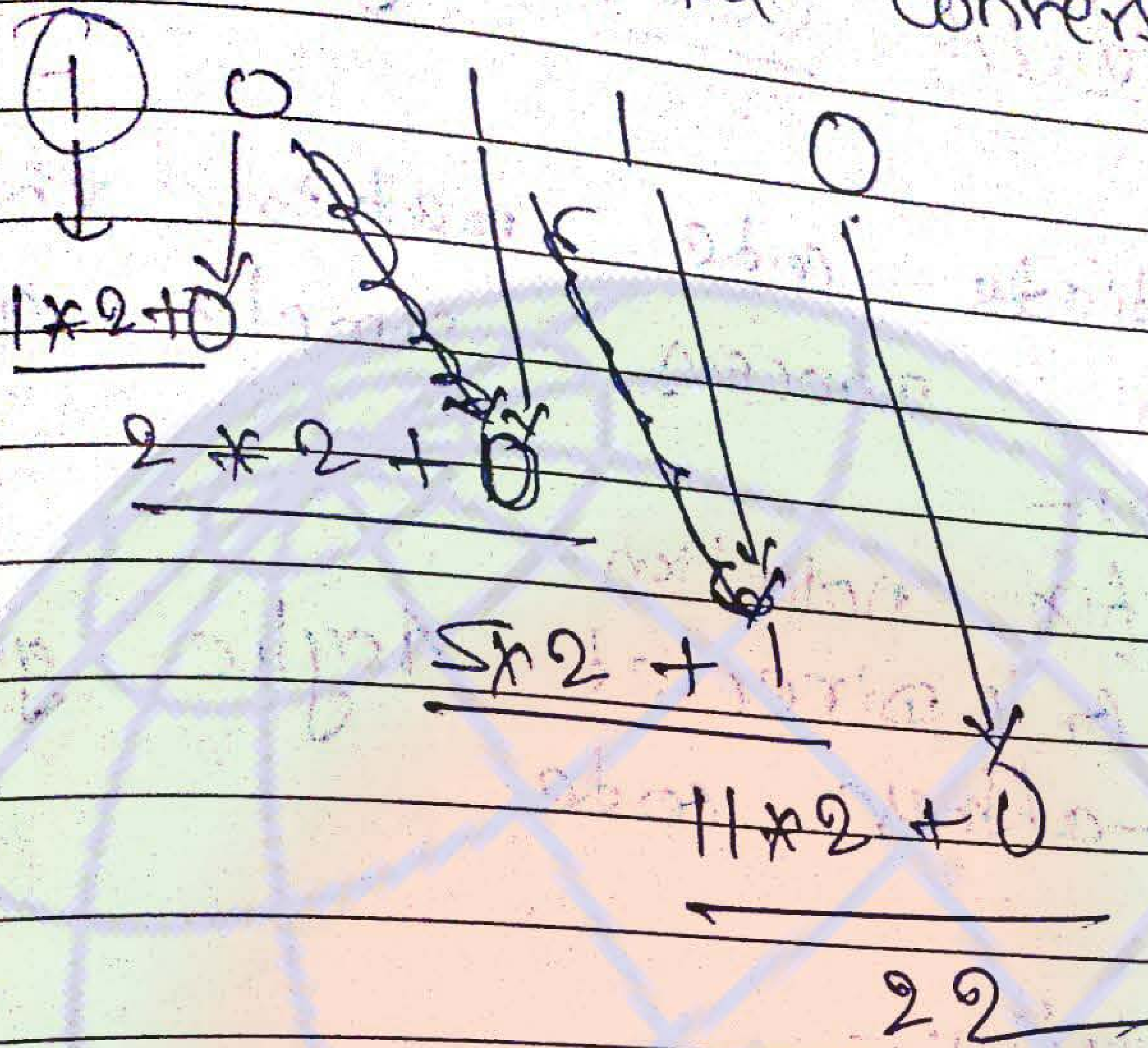


→ count no of bits

Scanned with CamScanner

WWW.GRADESETTER.COM

Binary to Decimal Conversion



N → L { N.val = L.val }

L → LB { L.val = L.val * 2 + B.val }
 | B { L.val = B.val }

B → 0 { B.val = 0 }
 | 1 { B.val = 1 }

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Intermediate code generation

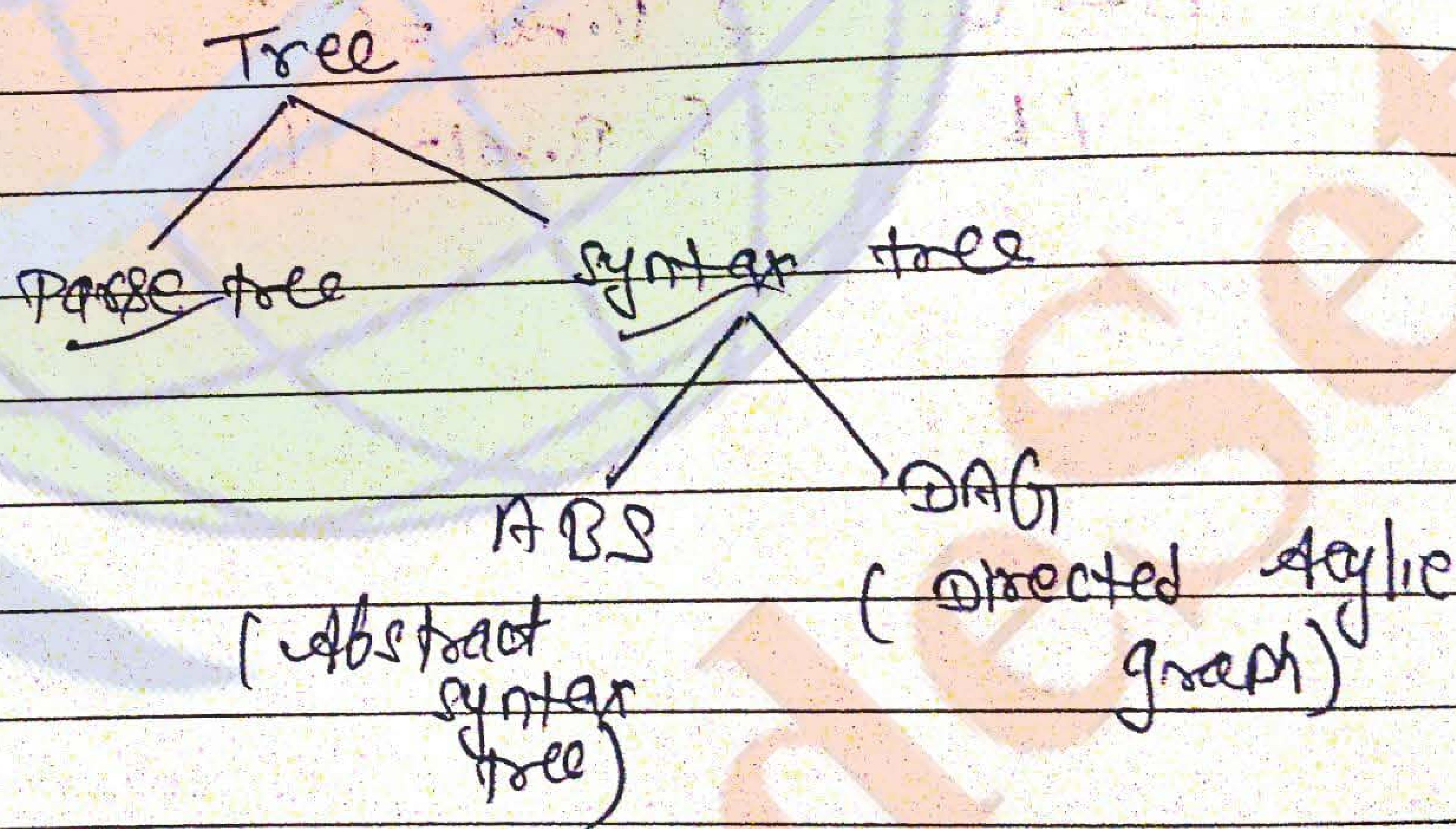
Intermediate code makes the code generation process simple and easy.

Representation:-

1) Postfix notation
2) DAG (Directed Acylic graph)
3) 3-address code

① Post-fix notation:-

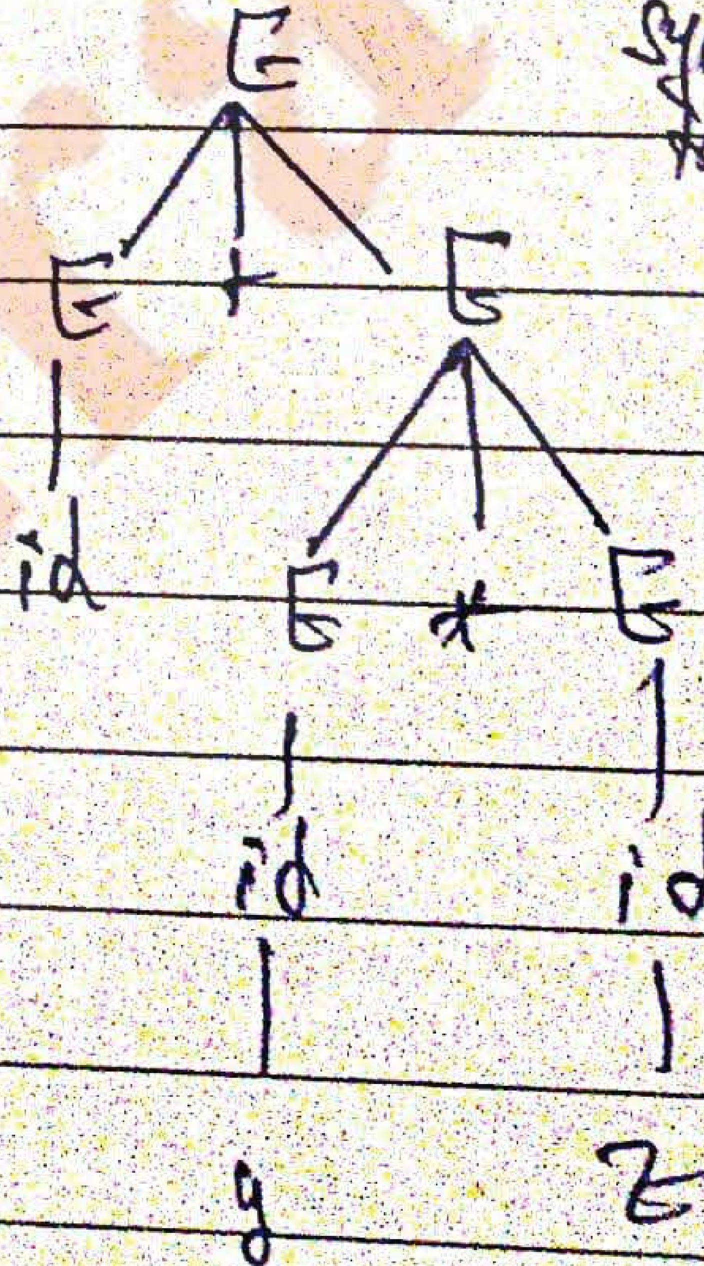
a+b : ab+
b*c : bc*
a+b*c : abc*+

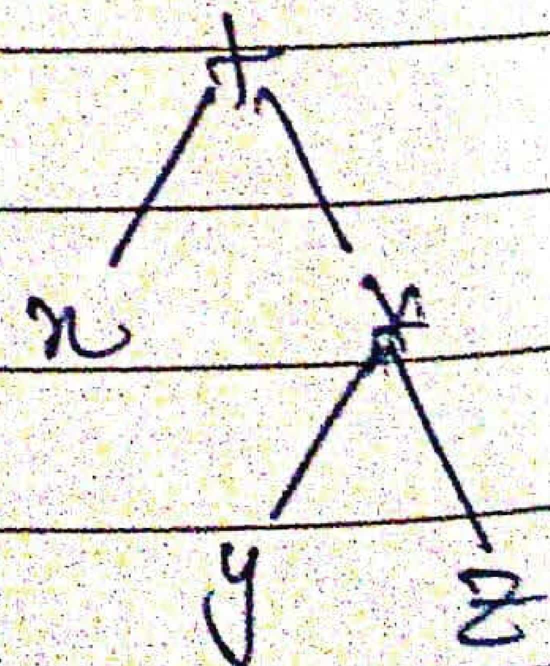


Parse & syntax tree

* { E → E+E
| E*E
| id }

Parse tree | Syntax tree-





Scanned with CamScanner

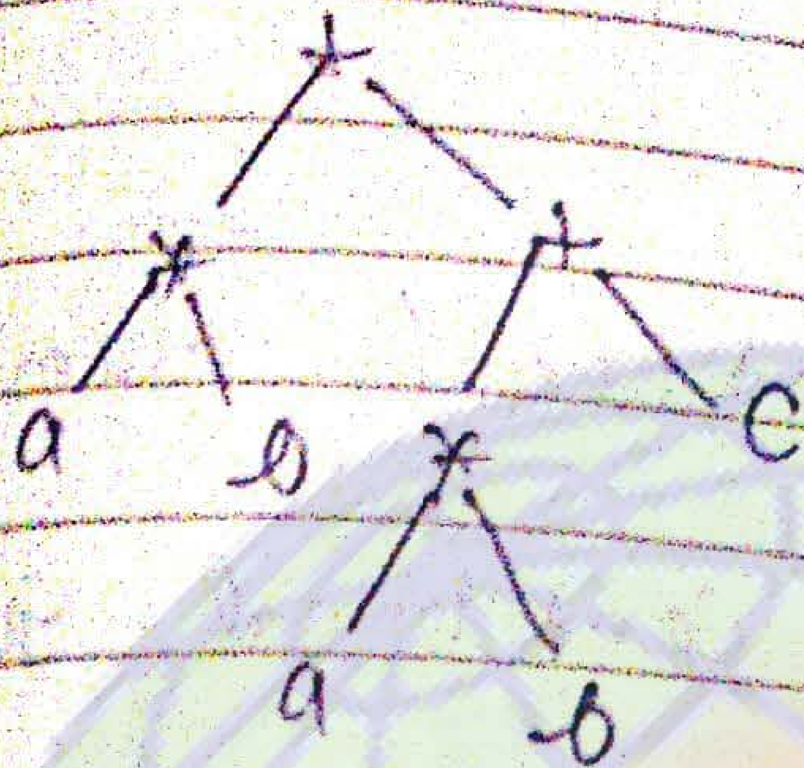
WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

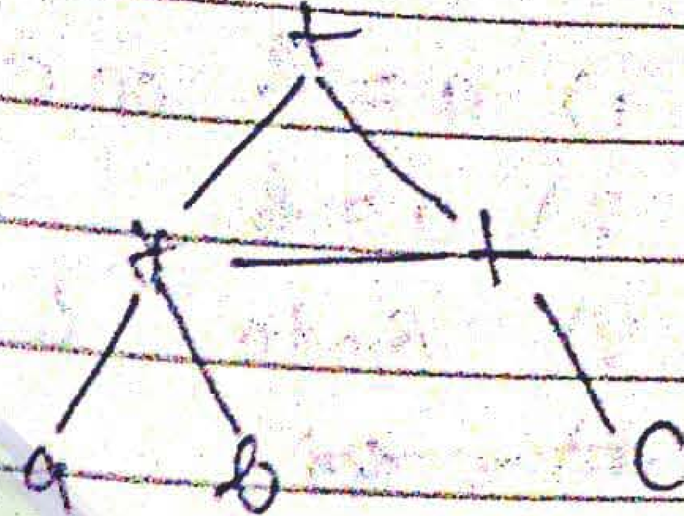
AST & DAG

* a*b + a*b+c

ASSADS: ↓

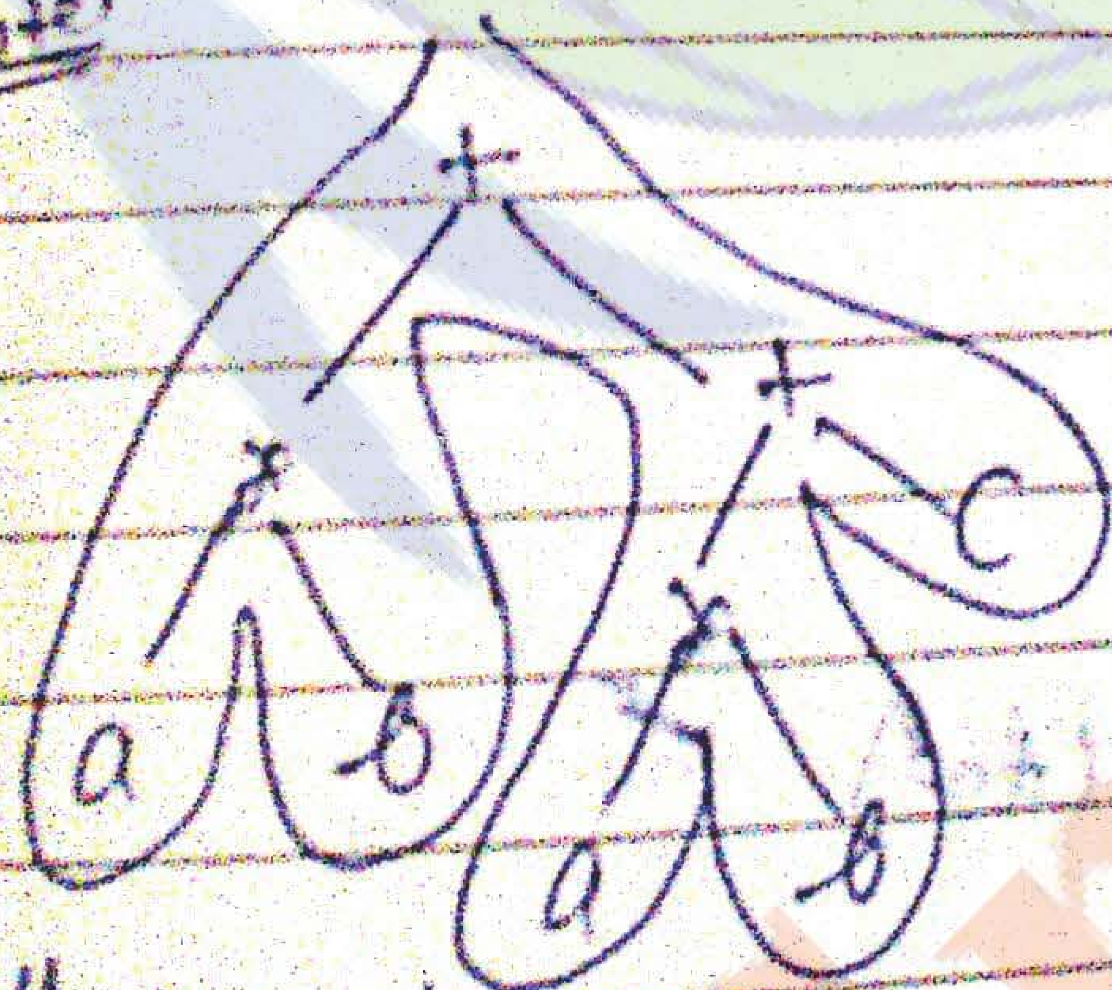


DAG: ↓



In DAG if a node is already created, insteas of creating the same node again, we will make use of already created node.

Note:



(2) 3-address code: In three-address code in any statement, maximum three temporary variable or three memory reference will be used. and atmost one operator can be used other then assignment operator.

Scanned with CamScanner

WWW.GRADESETTER.COM

a = b op c

Some valid three address code statements

1) a = b op c
2) a = b
3) Goto L — unconditional jump
4) If (rel) Goto L — conditional jump
5) a[i] = b
6) b = a[i]
7) a = *b

Q) Write the three address code for the following statements.

x = a + b + c * d where x, a are real; b, c, d are integer

t_1 = c * d
~~t_2 = a + b~~
~~x = t_1 + t_2~~
t_2 = b + t_1
t_3 = int to real (t_2)
x = a + t_3

Q) If (a > b)
a = a + 1
else
b = b + 1

Scanned with CamScanner

WWW.GRADESETTER.COM

Page No.
Date

~~1) If (a > b) Goto 2~~
~~2) a = a+1~~
~~3) If (b > a) Goto 4~~
~~4) b = b+1~~

```
If  a > b   goto L1
    b = b+1
   goto L2
L1 :   a = a+1
L2 :   ____
```

Backpatching:- Leaving the lables and filling them after is called backpatching.

Q) for (i=0 ; i ≤ n ; i++)

1st run

```
1) a=0
2) if i ≤ n
3) i = i+1
4) goto 2
```

reache

Q) for (i=0 ; i ≤ n ; i++)
{
a = a+1
}

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Page No.
Date

$i = 0$

$L1$: if $(i > n)$ goto $L2$

$a = a+1$

$i = i+1$

goto $L1$

$L2$:

Implementation of three-address code:-

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

manual representation:-

$t_1 = a+b$

$t_2 = -t_1$

$t_3 = c+d$

$t_4 = t_2 * t_3$

$t_5 = t_1 + c$

$t_6 = t_4 + t_5$

Now, In computer, there are three types of representation.

	Quadruple				Triple				Indirect triple	
temporary register	op	Arg1	Arg2	Result		op	Arg1	Arg2		
1	+	a	b	t_1	1)	+	a	b	(i)	1)
2	−	t_1		t_2	2)	−	(1)		(ii)	2)
3	+	c	d	t_3	3)	+	c	d	(iii)	3)
4	*	t_2	t_3	t_4	4)	*	(2)	(3)	(iv)	4)
5	+	t_1	c	t_5	5)	+	(1)	c	(v)	5)
6	+	t_4	t_5	t_6	6)	+	(4)	(5)	(vi)	6)

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

Dis:- more memory Adv: statements can be more around. (means eralech in any order)	Dis: statements can't be more around Adv: less memory	Dis: two time memory access Adv: statement's can be moves around.

☆ Code Optimization:-

It is proces of reducing the no of instruction without affecting the outcome of the source code.

It is of two types:-

(1) machine dependent code optimizatn-
It depend's on characterstics of the target machine for the addressing modes uses.
this can be achieves by, allocating sufficient no. of resources.

2) machine Independent code optim-
It is based on characterstics of the programming language. this can be achieves by analysing the code completly and finding alternative ~~sequence~~ equivalent of the source code.

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

* Techniques:

1) Common sub expression elimination:
It is a technique in which avoiding re-computation of the same expression.

$t_1 = 4*i$		$t_1 = 4*i$
$n = a[t_1]$		$n = a[t_1]$
$t_2 = 4*i$		$t_3 = 4*j$
$t_3 = 4*j$	optimized	$t_4 = a[t_3]$
$t_4 = a[t_3]$		$a[t_1] = t_4$
$a[t_2] = t_4$		$a[t_3] = n$
$t_5 = 4*j$		
$a[t_5] = n$		

2) Dead code elimination:-
A code is called dead code if it never get's executed

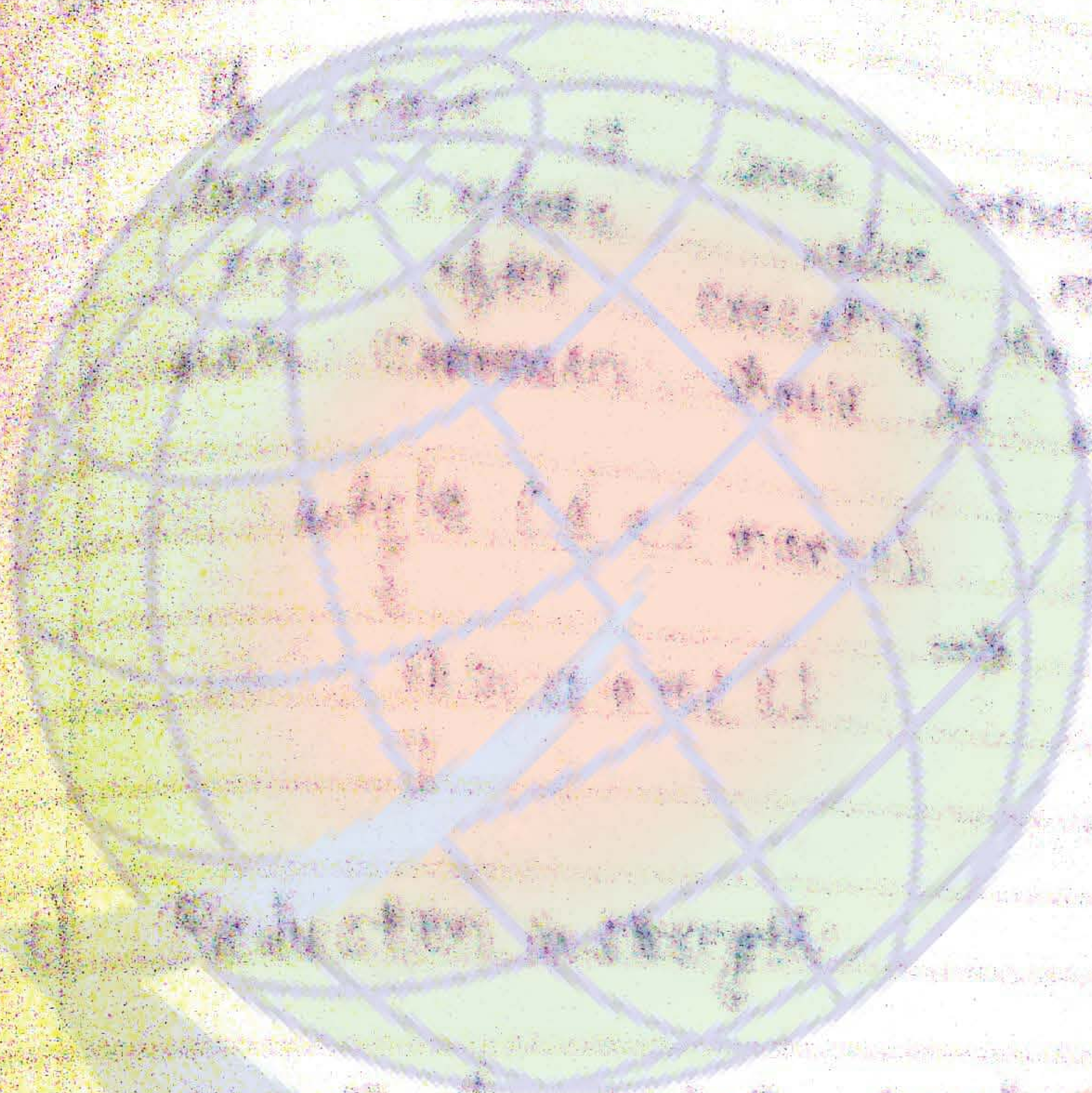
```
i=0
if (i>5)
{
  ____
  ____      } this is dead code
}
```

Here if statement is dead code which never satisfied.

3) Constant folding:- It is a technique in which constants expressions are calculated

Scanned with CamScanner
WWW.GRADESETTER.COM

WWW.GRADESETTER.COM



... remains unchanged.

Reduction in strength

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

during compilation time.

```
a = 3
b = 2+a;
c = 3*b+d        =>      a = 3
                         b = 5
                         c = 15+d
```

4) code motion:-

If there is some expression inside the loop, whose values remains unchanged even after executing the loops several times, such expression should be placed before the loop.

```
while (i <= max-1)              n = max - 1
{                       =>      while (i <= n)
   a = a + x[i]                 {
}                                  a = a + x[i]
                                }
```

5) Reduction in strength:-

strength of certain operator is more than the other operator

eg:

strength of * is more then +

```
                                a = 0
for (i=1; i<n; i++)             for (i=1; i<n; i++)
{                       =>      {
   a = i*5;                        a = a+5
}                               }
```

Scanned with CamScanner

WWW.GRADESETTER.COM

WWW.GRADESETTER.COM

6) Loop Invariant:-

Avoiding the evaluation or computation inside the loop.

```
for (i=0; i<n ; i++)          =>     n = a/b
{                                    for (i=1; i<n; i++)
  k = i + a/b                        {
}                                      k = i+n
                                     }
```

7) Loop fusion:-

merging several loops into one loop.

```
for (i=1; i<=10; i++)         =>     for (i=1; i<=100; i++)
{                                    {
  for (j=1; j<=10; j++)                printf ("Hi");
  {                                  }
    printf ("Hi");
  }
}
```

8) Loop Unrolling:-

No of jumps and tests will be reduced by writing the code two time.

```
while (i<=100)                =>     while (i<=100)
{                                    {
  a[i] = b[i]                          a[i] = b[i]
  i++                                  i++
}                                      a[i] = b[i]
                                       i++
                                     }
```

Scanned with CamScanner

WWW.GRADESETTER.COM