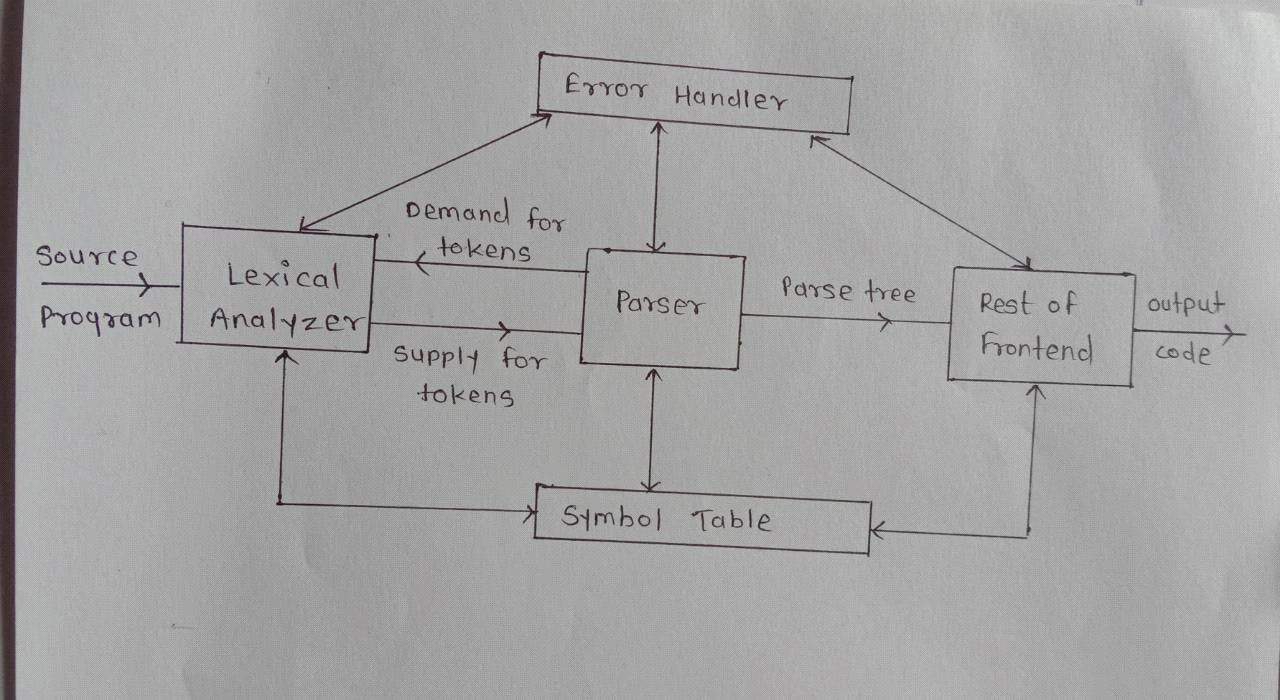
**Syntax Analyzer:-**

* Syntax analysis is the second phase in compilation.
* The syntax analyzer is called as parser.
* Parser basically checks for the syntax of the language.
* A syntax analyzer takes the tokens from the lexical analyzer & groups them in such a way that some programming structure(syntax) can be recognized.
* After grouping, the tokens if at all, any syntax can't be recognized then syntactic error will be generated.
* This overall process is called syntax checking of the language.



**Fig.Role of Parser**

* In the process of compilation the parser & lexical analyzer work together. i.e. When parser requires string of tokens it invokes lexical analyzer.
* The parser collects sufficient number of tokens & builds a parse tree.

ex. a= b+10;

* The above programming statement is first given to lexical analyzer. The lexical analyzer will divide it into group of tokens. The syntax analyzer takes the tokens as input & generates a tree like structure called parse tree.

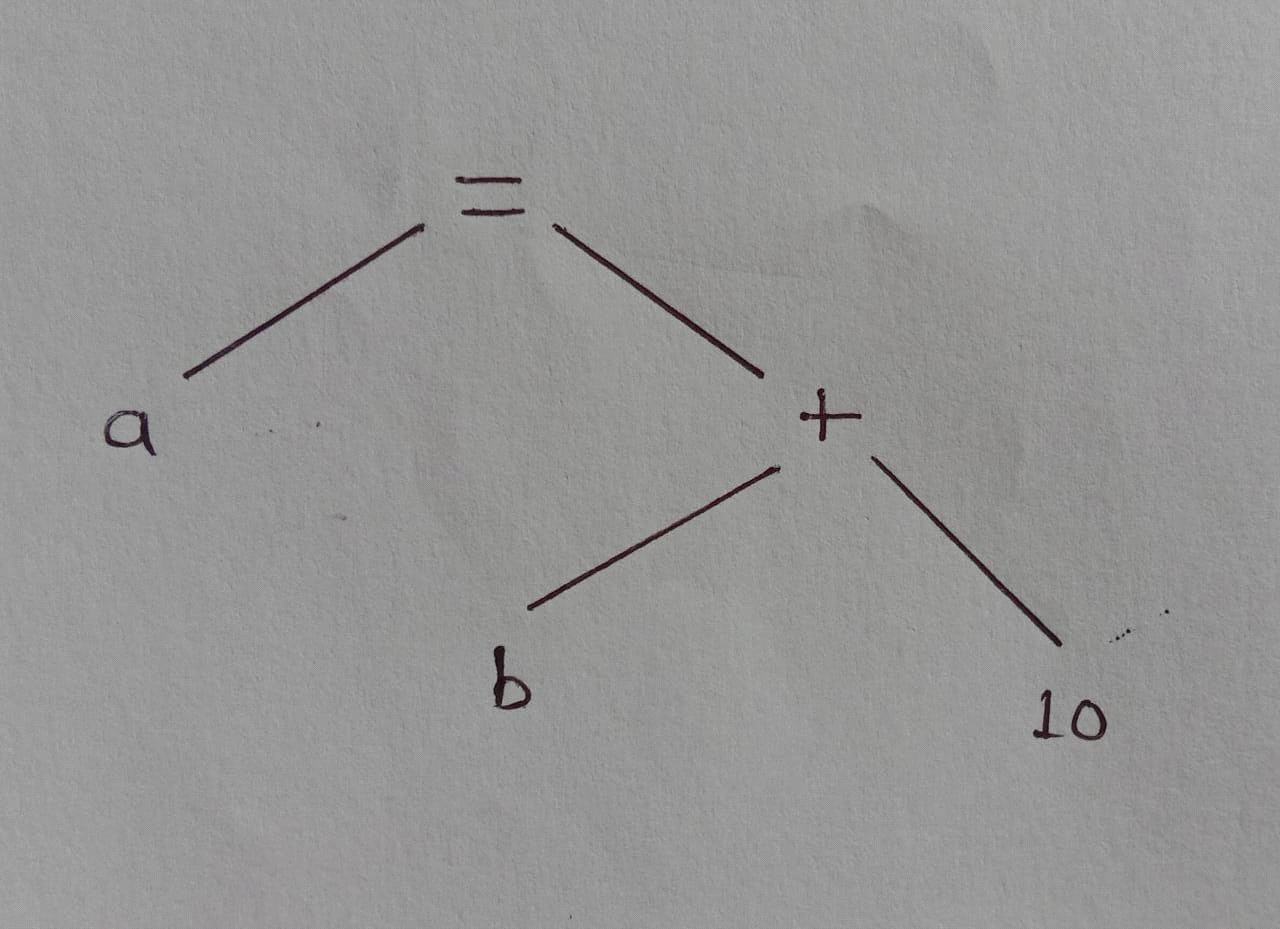


fig. parse tree for a= b+10;

**Issues in Parsing**

The issues in parsing are as follows:

* Specification of syntax
* Representation of input after parsing
* Parsing algorithm

1) Specification of Syntax:-

Specification of syntax means how to write any programming statement. There are certain characteristics of specification of syntax.

a) Specification should be precise & unambiguous.

b) Specification should be in detail.

c) Specification should be complete.

such specification is called context free grammar.

2) Representation of input after parsing:-

All the subsequent phases of compiler take the information from the parse tree being generated.

Information suggested by any input programming should not be differed after building the syntax tree for it.

3) Parsing algorithm:-

The parsing algorithm based on which we get the parse tree for the given input.

**Bottom-Up Parse**

-In bottom-up parsing method, the input string is taken first and we try to reduce this string with the help of grammar and try to obtain the start symbol.

-The process of parsing halts successfully as soon as we reach to start symbol.

-The parse tree is constructed from bottom to up that is from leaves to root.

-The input string is placed at the leaf nodes.

-The bottom-up parse tree is created starting from leaves, the leaf nodes together are reduced further to internal nodes, these internal nodes are further reduced and eventually a root node is obtained.

-In bottom-up parsing process the parser basically tries to identify R.H.S of production rule and replace it by corresponding L.H.S this activity is called reduction.

-In bottom-up parsing construction process indicates that the tracing of derivations are to be done in reverse order.

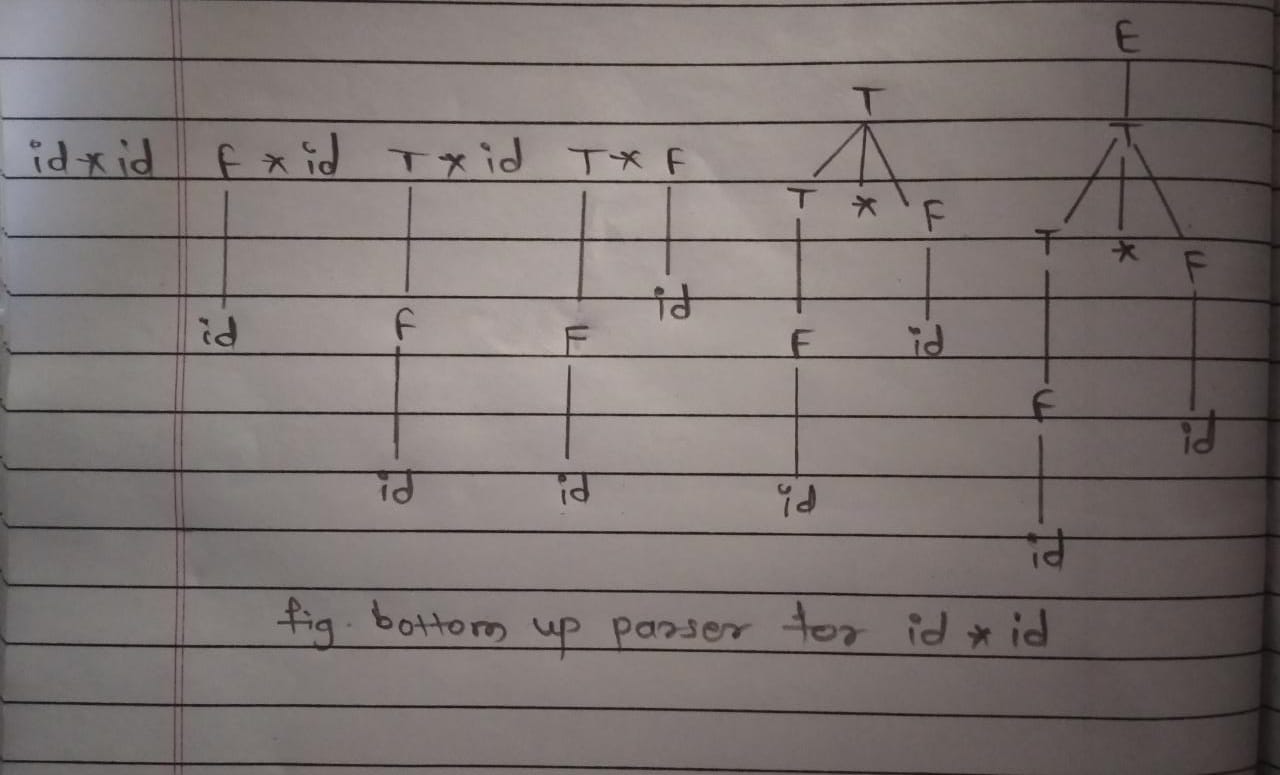
Ex: Consider the grammar

E->E+T|T

T->T\*F|F

F->(E)|id

The input string id\*id.

Construct bottom-up parser

**Handle**-

-A substring that could be reduced by appropriate non terminal, such a substring is called handle.

- Handle of right sentential from ϒ is a production A->β and a position of ϒ where the string β may be found and replaced by A to produce the previous right sentential form in rightmost derivation of ϒ.

example:- Consider the grammar

E->E+E

E->id

Now, consider the string id+id+id and rightmost derivation is

E->E+E

->E+E+E

->E+E+id

->E+id+id

->id+id+id

The underlined strings are called handles.

Example:S->aABe

A->Abc|b

B->d

S->aABe->aAde->aAbcde->abbcde

It follows that,

S->aABe is a handle of aABe in location 1.

B->d is a handle of aAde,in location 3.

A->Abc is a handle of aAbcde in location 2.

A->b is a handle of abbcde in location 2.

**Handle Pruning-**

-The process of discovering a handle and reducing it to the appropriate left-hand side is called handle pruning.

-Handle pruning forms the basic for a bottom up parsing method.

**Algorithm for handle pruning**

1. Start from ϒn, find a handle An->βn in ϒn, and replace βn by An to get ϒn-1.
2. Then find a handle An-1->βn-1 in ϒn-1 and replace βn-1 by An-1 to get ϒn-2.
3. Repeat this, until we reach S.

**Shift Reduce Parser**

-Shift reduce parser is worked on the principle of bottom up parser as like the bottom up parser it constructs parse tree from leaves to root.

-Shift reduce parser requires following data structures.

1)Input buffer storing the input string.

2)A stack for storing and accessing LHS and RHS of rule.

Initially both stack and input buffer is as follows.

W $

$ S

Stack Input Buffer

-The parser performs following basic operations

**1)Shift-**

Moving of the symbols from input buffer onto the stack, this action is called shift.

**2)Reduce-**

If the handle appears on the top of the stack then reduction of it by appropriate production is done. That means RHS of rule is popped of and LHS is pushed in. This action is called Reduce action.

**3)Accept-**

If the stack contains start symbols only and input buffer is empty at the same time that then that action is called accept. When accept state is obtained in the process of parsing then it means a successful parsing is done.

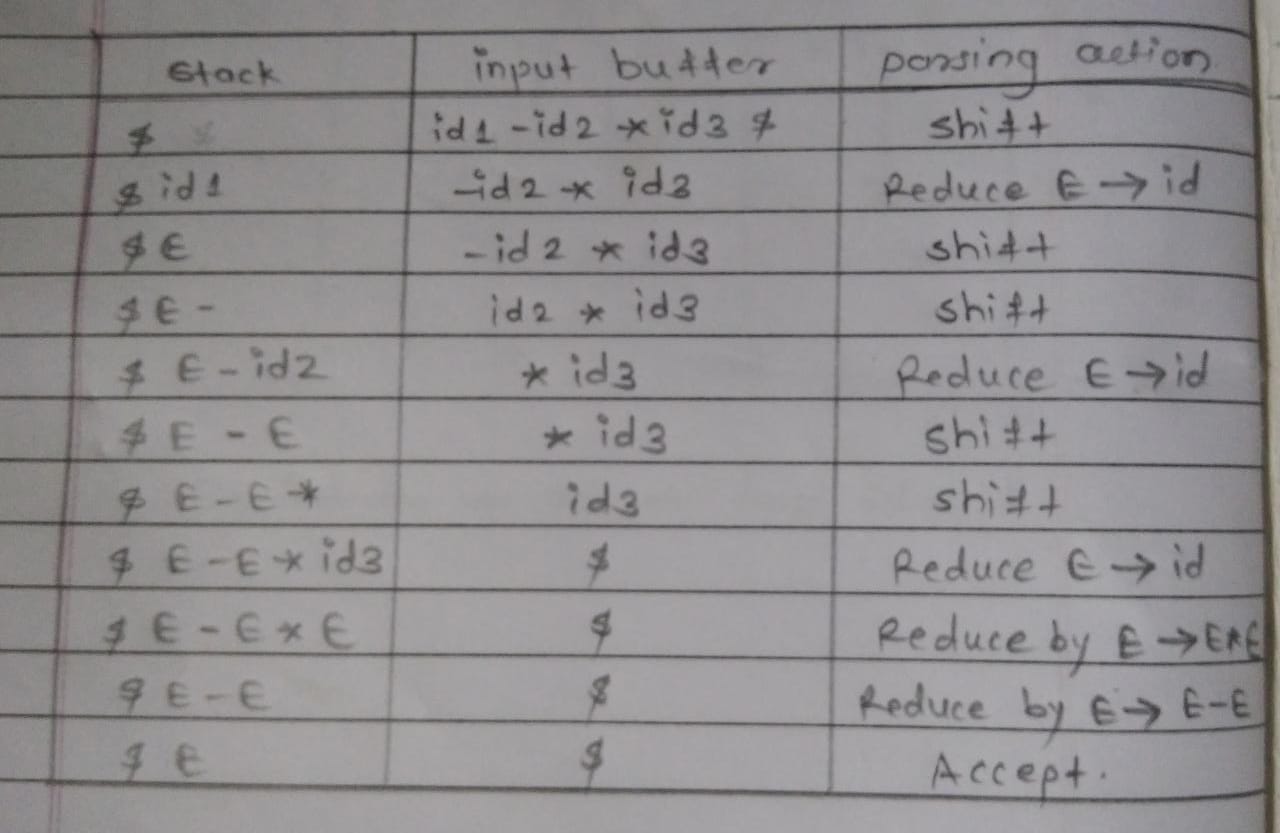
**4)Error-**

A situation in which parser can’t either shift or reduce the symbols, it can’t even perform the accept action is called as error.

Ex. 1.Consider the grammar

E->E-E|E\*E|id

Perform shift-reduce parsing of the input string id1-id2\*id3



1)If incoming operator has more priority that in stack operator then perform stack.

2)If in stack operator has same or less priority than the priority of incoming operator then perform reduce.

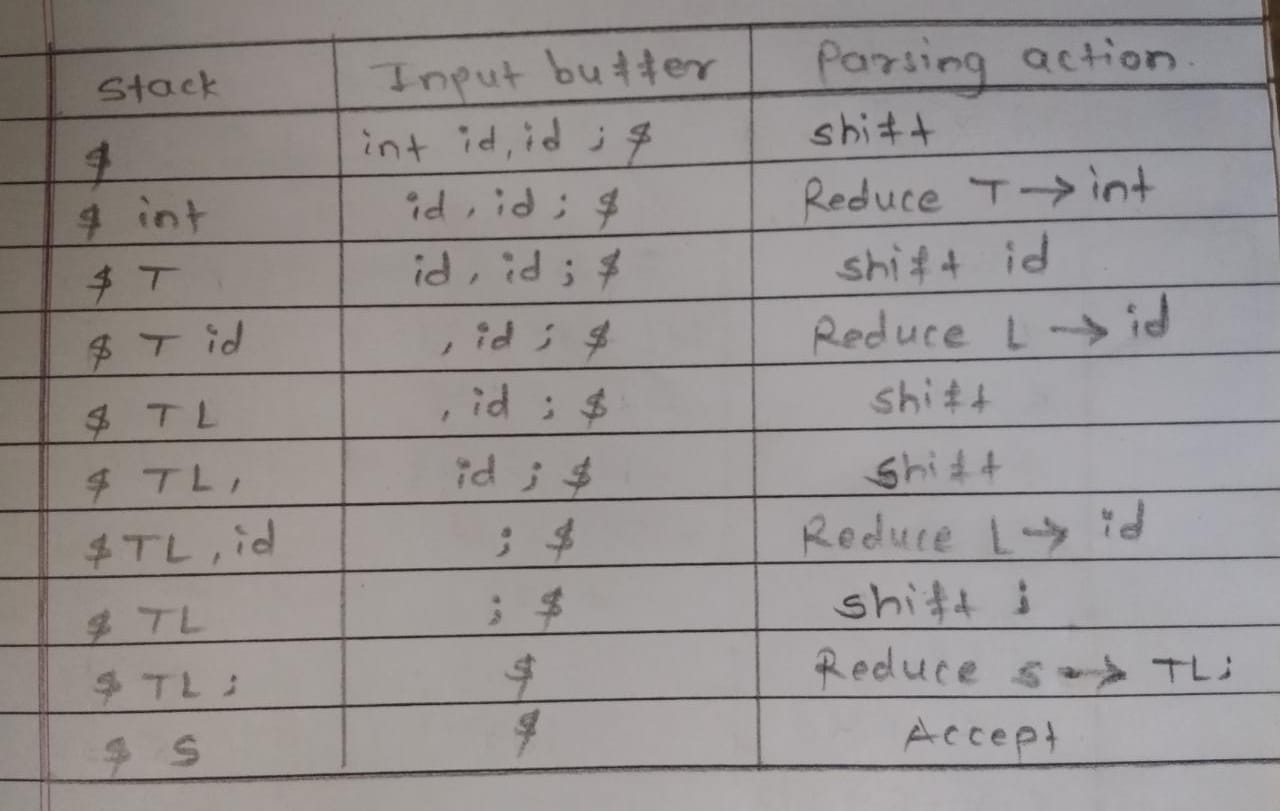
2.Consider the grammar

S->TL;

T->int |float

L->L ,id |id

Parse the input string int id, id ;using shift reduce.

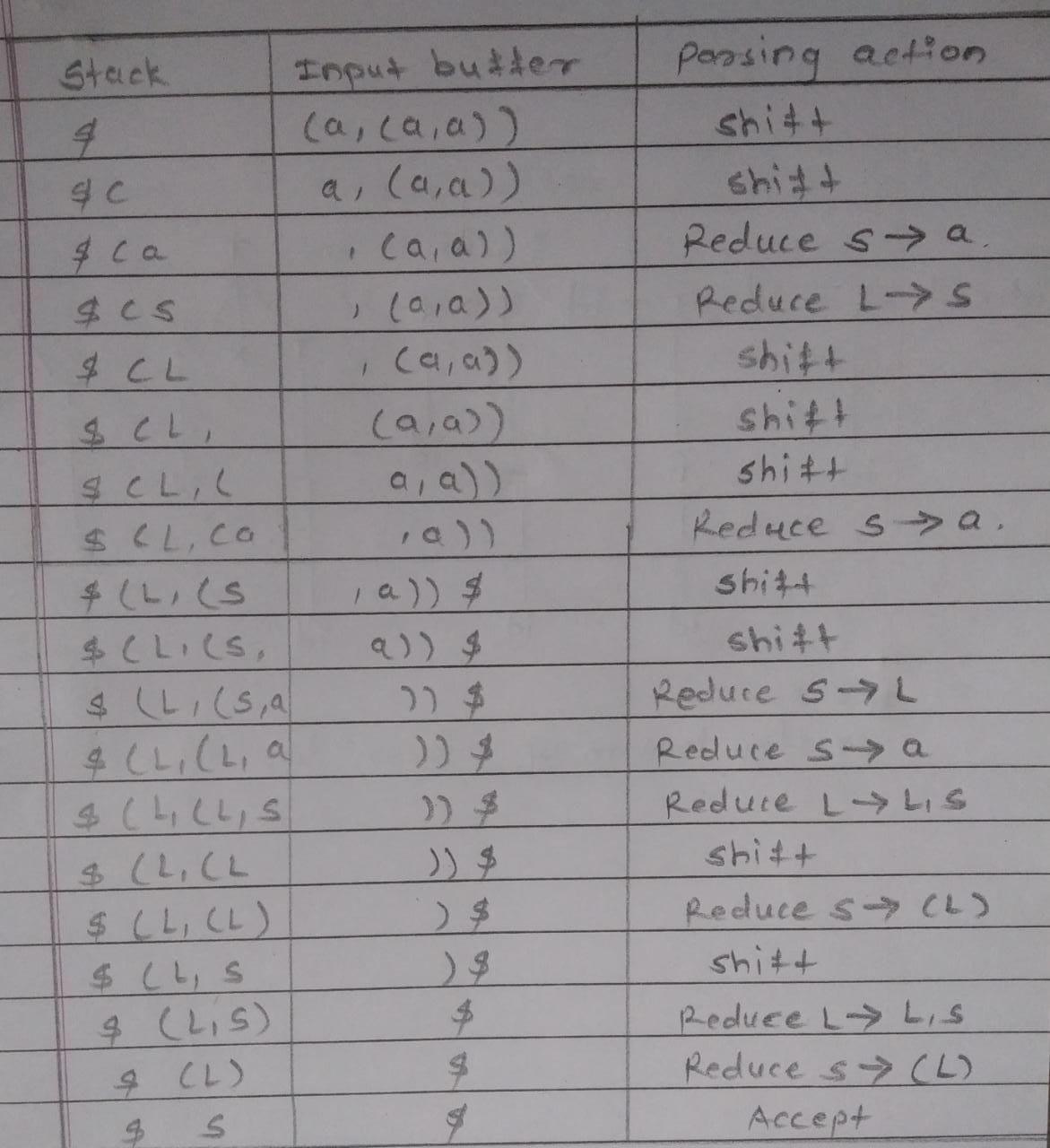


3.Consider the following grammar

S->(L)|a

L->L,S|s

Parse the input string (a,(a, a)) using shift reduce parser.



**Conflicts during Shift-Reducing parsing**

-There are context free grammars for which shift reduce parser can’t be used.

-Stack contents and the next input symbol may not decide action.

1)shift/reduce conflict-

Whether make a shift operation or a reduction.

2)reduce/reduce conflict-

The parser can’t decide which of several reductions to make.

**LR Parser-**

-LR parser is the most efficient methods of bottom-up parsing which can be used to parse the large class of context free grammars .

-This method is called LR(k)parsing here L stands for left to right scanning,

R stands for rightmost derivation in reverse, k is number of input symbols. When k is omitted k is assumed to be 1.

**Properties/Advantages of LR parser-**

-This method is non backtracking shift reduce parsing .

-All class of grammars can be parse using LR methods for parsing .

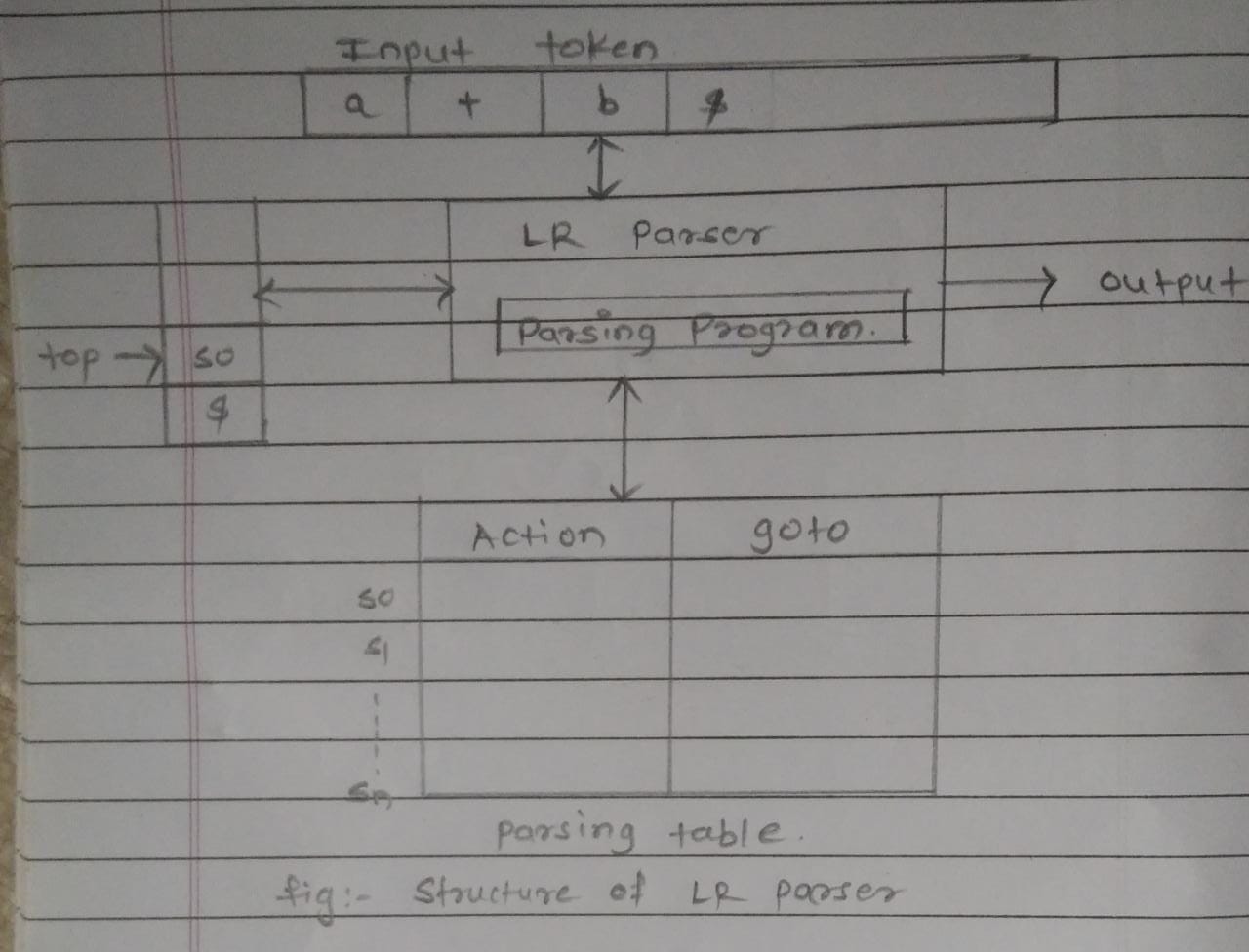
-It is superset of predictive parser.

-Syntax error are detected immediately during scanning of source code.

-It accepts left recursive grammar also.

-It is almost used in compilers of all programming language.

**Structure of LR parser-**



The LR parser consists of input buffer for storing the input string.

A stack for storing the grammar symbols, output.

Parsing table comprised of two parts namely action and goto.

**Algorithm for LR parser**-

1)It initializes the stack with start symbol and invokes scanner to get next token.

2)It determines Sj the state currently on the top of the stack and ai the current input symbol.

3)It consults the parsing table for the action[Sj,ai] which can have one of the following values.

i)Si means shift state i.

ii)rj means reduce by rule j.

iii)Accept means successful parsing is done.

iv)Error indicates syntactical error.

**Types of LR parser**-

The types of LR parser are as follows.

LR Parser

Canonical LR Parser

LALR Parser

SLR Parser

**SLR Parser-**

-The simplest form of LR parsing called SLR parser. This parser is the weakest than the other parser but easy to implement. The parsing can be done as follow

Context free grammar

Construction of canonical set of items

Output

Input string

Parsing of input String

Construction of SLR parsing table

**Definitions-**

**1)LR(0) item-**

An LR(0)item is a production with special marker(**·**)marking a position within the string on the right side of the production.

If the production is S→ABC

Then the possible LR(0)items are

S→·ABC

S→A·BC

S→AB·C

S→ABC·

The production S→∊ generates only one item

S→·

**2)Augmented grammar-**

If a grammar G having start symbol S then augmented grammar is a new grammar G1 in which S1 is new start symbol such that S1→S. The purpose of this grammar is to indicate acceptance of input. That is when parser is about to reduce S1→S it reaches to acceptance state.

**3)Kernel item/Non Kernel item-**

The collection of items S1→·S and all the items whose dot are not at the leftmost end of RHS of the rule.

Non Kernel items are the items which have the dot at leftmost end.

**4)Viable Prefix-**

It is the set of prefixes in the right sentential form of production A→α.This set can appear on the stack during shift/reduce action.

**Closure Function-**

If I is a set of items for grammar G, then set of items CLOSURE(I) is constructed from I by the rules.

Every item in I, If ·appears just before any nonterminal then all rules of nonterminal are also added in that item. It is explained in following example.

1)If production rule A→α·Bβ is present in solution item I then CLOSURE(I) function is executed means if B→ɣ is production then it adds B→·ɣ into that item I, if it is not there.

2)This is applicable for all B-productions.

**GOTO Operation-**

The function goto can be defines as follows.

If there is a production A→αB·β then (A→α.Bβ,B)=A→αB·β.That means simply shifting of · one position ahead over the grammar symbol(may be terminal or non terminal). The rule A→α·Bβ is in I then the same goto function can be written as goto(I,B).

**Procedure of finding out SLR parse table-**

1)Find out augmented grammar of that grammar.

2)Find out LR(0)Items like I0,I1,I2,I3 using closure function and GOTO function, which is explained as follows.

3)I0 Items can be find out using following rules

a)Add item S1→·S

b)Apply closure operation on S.

4)I1,I2,………….,In items can be find out by using GOTO and closure function.

5)Draw DFA for the above items.

6)Construct SLR Parse table using following rules

a)Find out FOLLOW of all non-terminals

b)Divide parse table into two parts that is Action part and GOTO part.

c)Assume there are total n items that are I0,I1,………,In. There are total n+1 rows of Acton table as well as GOTO table.

d)All terminals are column headers of the Action table and all non-terminals are column headers of GOTO table.

e)If item A→α·aβ is any In, and GOTO(In,a)=In then add shift m in to Action[n,a].

g)If item A→α· is any In, then reduce A→α to Action [n,a] for all a in FOLLOW(A) i.e. Action[n,a]=rm where mth production rule is A→α used for reduction.

h)If S1→S· is any In then add Accept in Action[n,$] i.e. Action[n,$]=Accept.

i)If there is GOTO FUNCTION like GOTO[In,A]=Im then add GOTO[n,A]=m

j)All entries which are not defined make them as error.

If once cell contain only one entry then it is called SLR grammar.

**1)Find out SLR parse table for the following grammar**

**E→E+T|T**

**T→T\*F|F**

**F→(E)|id**

**Answer**:

The augmented grammar is as follows.

R0 E’→E

R1 E→E+T

R2 E→T

R3 T→T\*F

R4 T→F

R5 F→(E)

R6 F→id

In this grammar R0 represented rule number 0, similarly R6 represents number 6. It is mentioned in table for reducing purpose. Now, find out LR(0) sets of item.

Find out I0 items.

I0→E’→·E Now · is just before E so closure function is operated on E&E-productions are added.

E→·E+T

E→·T

T→·T\*F Now · is just before T so closure function is operated on T & T productions are added.

F→·(E) Now · is just before F so closure function is operated on F and F productions are added.

F→·id

Now, find out I1 item using goto function i.e. GOTO(I0,E)=I1

I1:E’→E·

E→E·+T GOTO function is operated on every ·E of I0 in I1 · operator is just shifted after E.

Now, find out I2 item using goto function i.e. GOTO(I0,T)=I2

I2:E→T·

T→T·\*F

GOTO function is operated on every ·T of I0 in I2 · operator is just shifted after T.

Now, find out I3 item using goto function i.e. GOTO(I0,F)=I3

I3:T→F· GOTO function is operated on every ·F of I0 in I3 · operator is just shifted after F.

Now, find out I4 item using GOTO function i.e. GOTO(I0,C)=I4

I4:F→(·E) GOTO function is operated on every ·F of I0 E-productions are added.

E→·E+T

E→·T

T→·T\*F

T→·F

F→·(E)

F→·id

Now, find out I5 item using GOTO function i.e. GOTO(I0,id)=I5

I5:F→id· GOTO function is operated on every ·id of I0

Now, find out I6 item using GOTO function i.e. GOTO(I1,t)=I6

I6:E+·T

T→·T\*F

T→·F

F→·(E)

F→.id

GOTO function is operated on every ·t of I1, · is just shifted after t. Now, ·is just before T so closure function is operated on T. E-productions are added.

Now, find out I7 item using GOTO function i.e. GOTO(I2,\*)=I7

I7: T→T\*·F

F→·(E)

F→·id

Now, find out I8 item using GOTO function i.e. GOTO(I4,E)=I8

I8:F→(E·)

E→E·+T

Now, using GOTO function i.e. GOTO(I4,T)=I2

I2:E→T·

T→T·\*F

Now using GOTO function i.e. GOTO (I4,F)=I3

Now using GOTO function i.e. GOTO (I4,C)=I4

Now using GOTO function i.e. GOTO (I4,id)=I5

Now using GOTO function i.e. GOTO (I6,T)=I9

I9:E+T·

T→T·\*F

Now using GOTO function i.e. GOTO(I6,F)=I3

Now using GOTO function i.e. GOTO (I6,C)=I4

Now using GOTO function i.e. GOTO (I6,id)=I5

Now using GOTO function i.e. GOTO (I7,F)=I10

I10:T→T\*F·

Now using GOTO function i.e. GOTO (I7,C)=I4

Now using GOTO function i.e. GOTO (I7,id)=I5

Now using GOTO function i.e. (I8,))=I11

I11:F→(E)·

Now using GOTO function i.e. GOTO (I8,t)=I6

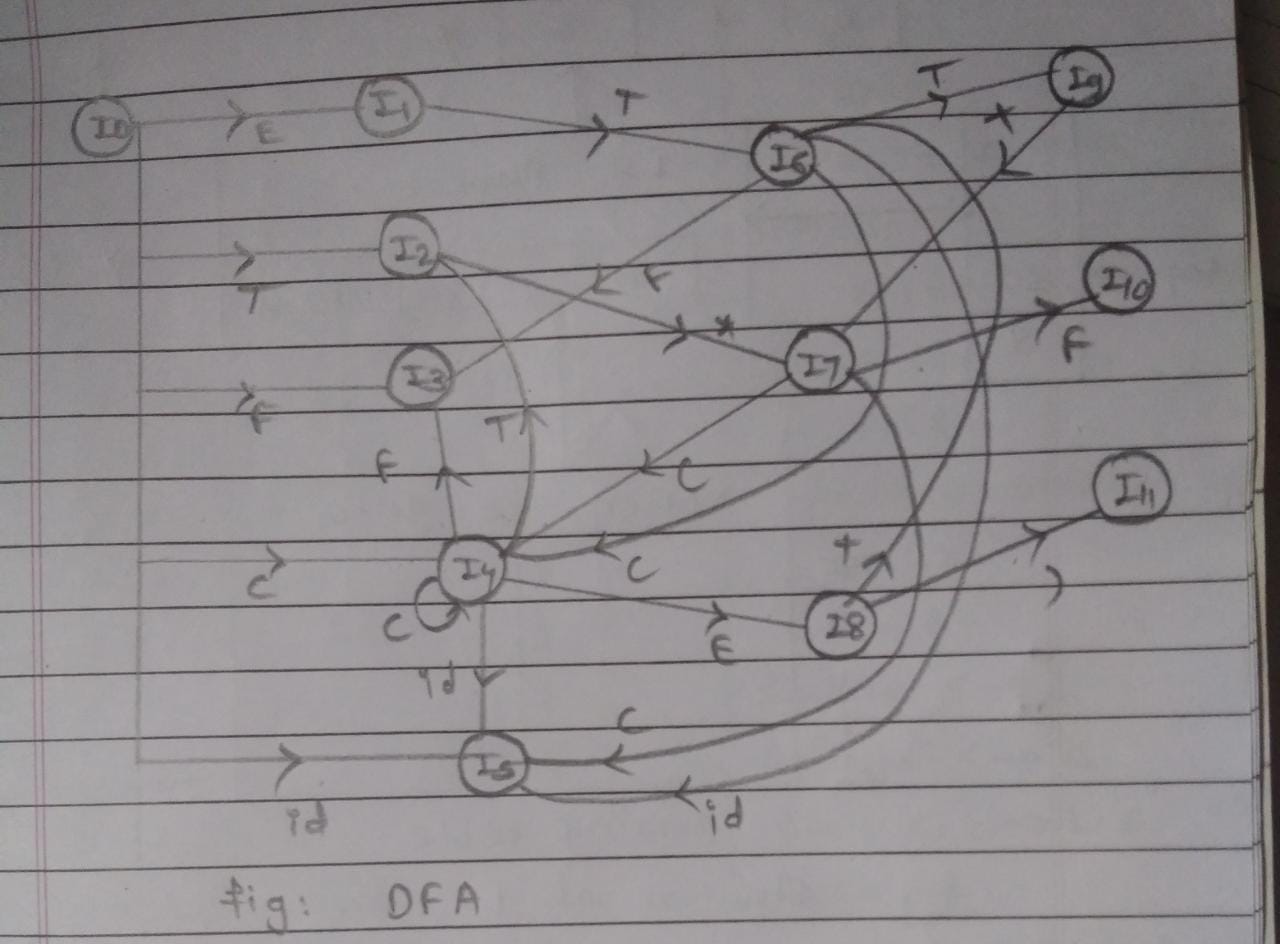
Now using GOTO function i.e. GOTO (I9,\*)=I7

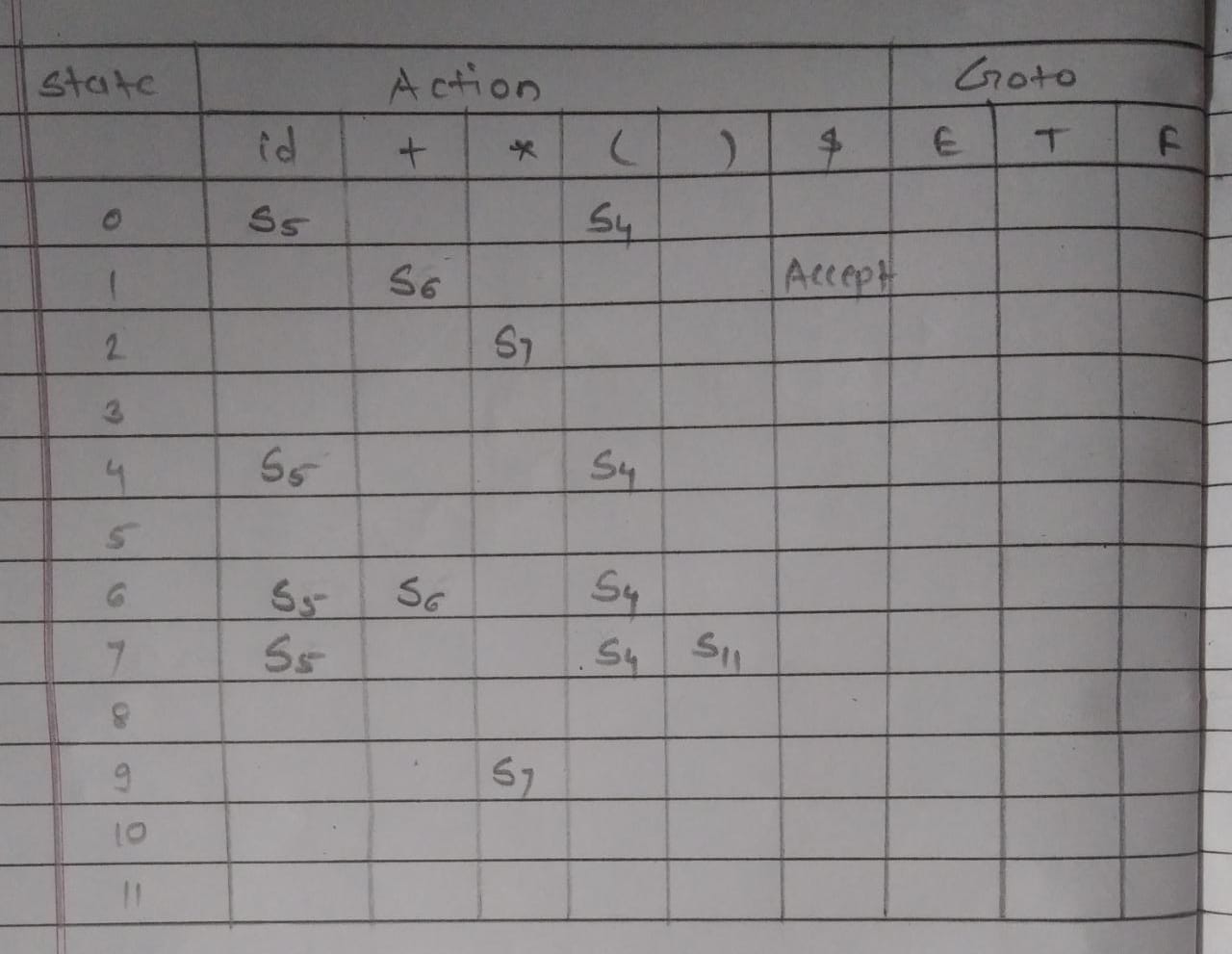
FOLLOW of non-terminal is

FOLLOW(E)={$,t,)}

FOLLOW(T)={$,t,\*,)}

FOLLOW(F)={$,t,\*,)}





**Check the following grammar is SLR1 grammar or not?**

**S→A|D**

**A→B;C**

**B→bd|B;d**

**C→ae|a;C**

**Answer**

No D is useless symbol it is deleted, then grammar is

S→A

A→B;C

B→bd|B;d

C→ae|a;C

The augmented grammar for the above grammar is

R0 S’→S

R1 S→A

R2 A→B;C

R3 B→bd

R4 B→B;d

R5 C→ae

R6 C→a;C

Now, we find LR(0) items,

I0:S’→·S

S→·A

A→·B;C

B→·bd

B→·B;d

GOTO(I0,S)=I1

I1:S’→S·

GOTO(I0,A)=I2

I2:S→A·

GOTO(I0,B)=I3

I3:A→B·;C

B→B·;d

GOTO(I0,b)=I4

I4:B→b·d

GOTO(I3,;)=I5

I5:A→B;·C

C→·ae

C→·a;C

B→b;·d

GOTO(I4,d)=I6

I6:B→bd·

GOTO(I5,C)=I7

I7:A→B;C·

GOTO(I5,a)=I8

I8:C→a·e

C→a·;C

GOTO(I5,d)=I9

I9:B→b;d·

GOTO(I8,e)=I10

I10:C→ae·

GOTO(I8,;)=I11

I11:C→a;·C

C→·ae

C→a;C

GOTO(I11,C)=I12

I12:C→a;C·

GOTO(I11,a)=I13

I13:C→a·e

C→a·;C

GOTO(I13,e)=I10

GOTO(I13,;)=I11

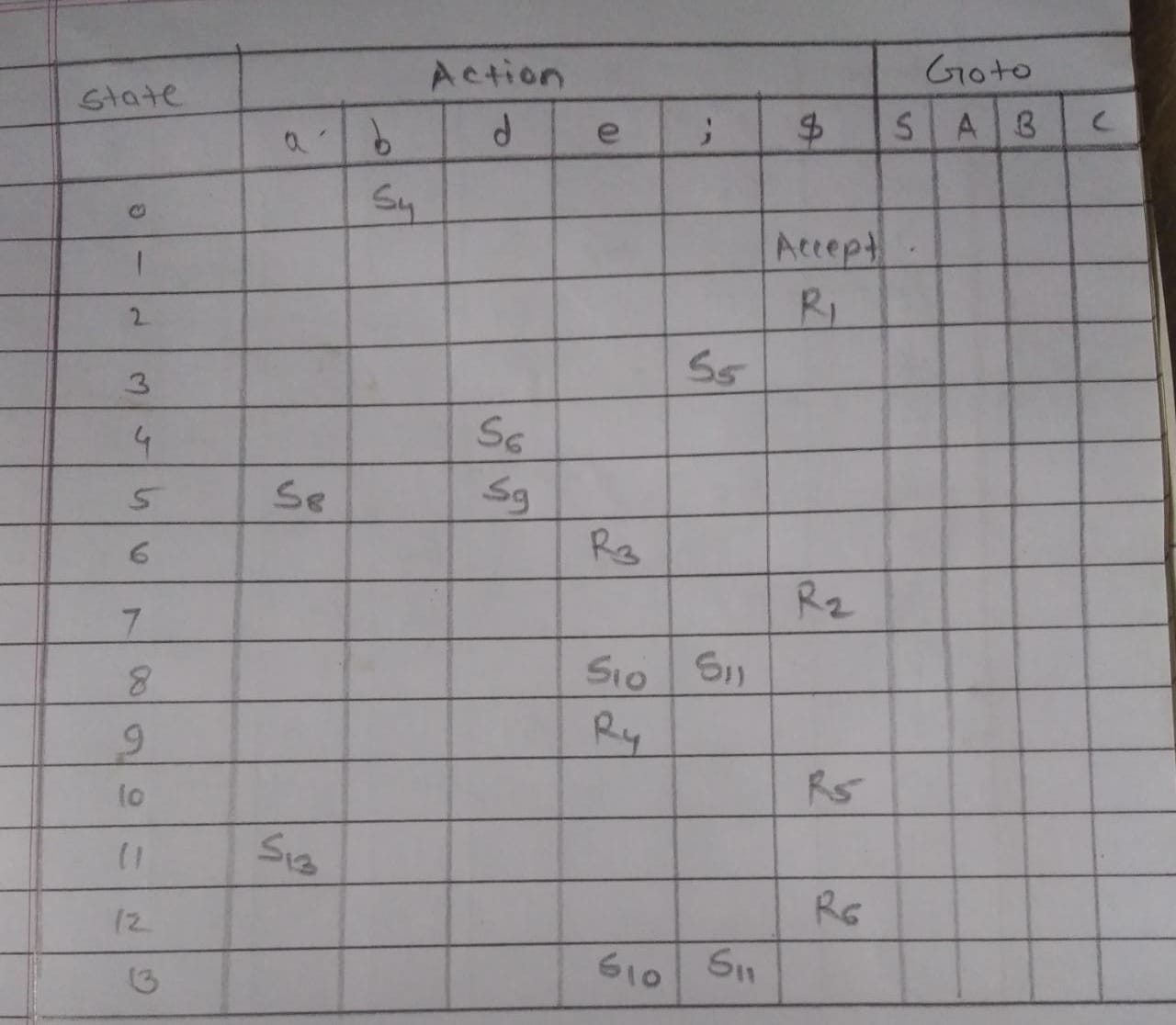
FOLLLOW(S)={$}

FOLLOW(A)={$}

FOLLOW(B)={$}

FOLLOW(C)={$}

SLR parse table is shown in following figure.



E’->E R0

E->E+T R1

E->T R2

T->T\*F R3

T->F R4

F->(E ) R5

F->id R6

I0:

E’->.E

E->.E+T

E->.T

T->.T\*F

T->.F

F->.(E )

F->.id

Now we can find the item set I 1 Goto(I0,E)=I1

I1 :

E’->E.

E->E.+T

Now we can find the item set I 2 Goto(I0,T)=I2

I2 :

E->T.

T->T.\*F

Now we can find the item set I 3 Goto(I0,F)=I3

I3 :

T->F.

Now we can find the item set I 4 Goto(I0,()=I4

I4 :

F->(.E)

E->.E+T

E->.T

T->.T\*F

T->.F

F->.(E )

F->.id

Now we can find the item set I 5 Goto(I0,id)=I5

I5 : F->id.

Now we can find the item set I 6 Goto(I1,+)=I6

I6 :

E->E+.T

T->.T\*F

T->.F

F->.(E )

F->.id

Now we can find the item set I 7 Goto(I2,\*)=I7

I7 :

T->T\*.F

F->.(E )

F->.id

Now we can find the item set I 8 Goto(I4,E)=I8

I8 :

F->(E.)

E->E.+T

Goto(I4,T)=I2

Goto(I4,F)=I3

Goto(I4,()=I4

Goto(I4,id)=I5

Now we can find the item set I 9 Goto(I6,T)=I9

I9 :  E->E+T.

T->T.\*F

Goto(I6,F)=I3

Goto(I6,()=I4

Goto(I6,id)=I5

Now we can find the item set I 10 Goto(I7,F)=I10

I10 :

T->T\*F.

Now we can find the item set I 11 Goto(I8,))=I11

I11 :

F->(E).

Goto(I8,+)=I6

Goto(I9,\*)=I7

|  |  |  |
| --- | --- | --- |
|  | First | Follow |
| E | {(,id} | {$,+,)} |
| T | {(,id} | {$,+,),\*} |
| F | {(,id} | {$,+,),\*} |

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

Consider the following grammar.

S->AB|gDa

A->ab|c

B->dC

C->gC|g

D->fD|g

Now we can write augumented grammar for the above grammar.

S’->S R0

S->AB R1

S->gDa R2

A->ab R3

A->c R4

B->dC R5

C->gC R6

C->g R7

D->fD R8

D->g R9

Now we can write the I0 item set

I0 :

S’->.S

S->.AB

S->.gDa

A->.ab

A->.c

Now we can find item set I1 Goto(I0,S)=I1

I1 :S’->S.

Now we can find item set I2 Goto(I0,A)=I2

S->A.B

B->.dC

Now we can find item set I3 Goto(I0,g)=I3

S->g.Da

D->.fD

D->.g

Now we can find item set I4 Goto(I0,a)=I4

A->a.b

Now we can find item set I5 Goto(I0,c)=I5

A->c.

Now we can find item set I6 Goto(I2,B)=I6

S->AB.

Now we can find item set I7 Goto(I2,d)=I7

B->d.C

C->.gC

C->.g

Now we can find item set I8 goto(I3,D)=I8

S->gD.a

Now we can find item set I9 Goto(I3,f)=I9

D->f.D

D->.fD

D->.g

Now we can find item set I10 Goto(I3,g)=I10

D->g.

Now we can find item set I11 Goto(I4,b)=I11

A->ab.

Now we can find item set I12 Goto(I7,C)=I12

B->dC.

Now we can find item set I13 Goto(I7,g)=I13

C->g.C

C->g.

C->.gC

C->.g

Now we can find item set I14 goto(I8,a)=I14

S->gDa.

Now we can find item set I15 goto(I9,D)=I15

D->fD.

Goto(I9,f)=I9

Goto(I9,g)=I10

Now we can find item set I16 goto(I13,C)=I16

C->gC.

Goto(I13,g)=I13

|  |  |  |
| --- | --- | --- |
|  | First | Follow |
| S | {a,c,g} | {$} |
| A | {a,c} | {d} |
| B | {d} | {$} |
| C | {g} | {$} |
| D | {f,g} | {a} |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Action | | | | | | | Goto | | | | |
|  | a | b | c | d | f | g | $ | S | A | B | C | D |
| 0 | S4 |  | S5 |  |  | S3 |  | 1 | 2 |  |  |  |
| 1 |  |  |  |  |  |  | Accept |  |  |  |  |  |
| 2 |  |  |  | S7 |  |  |  |  |  | 6 |  |  |
| 3 |  |  |  |  | S9 | S10 |  |  |  |  |  | 8 |
| 4 |  | S11 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  | R4 |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  | R1 |  |  |  |  |  |
| 7 |  |  |  |  |  | S13 |  |  |  |  | 12 |  |
| 8 | S14 |  |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  | S9 | S10 |  |  |  |  |  | 15 |
| 10 | R9 |  |  |  |  |  |  |  |  |  |  |  |
| 11 |  |  |  | R3 |  |  |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  | R5 |  |  |  |  |  |
| 13 |  |  |  |  |  | S13 | R7 |  |  |  | 16 |  |
| 14 |  |  |  |  |  |  | R2 |  |  |  |  |  |
| 15 | R8 |  |  |  |  |  |  |  |  |  |  |  |
| 16 |  |  |  |  |  |  | R6 |  |  |  |  |  |

Augumented grammar is

A->E

E->TE’

E’->+TE’

E’->€

T->( E )

T->id

I0:

A->.E

E->.TE’

T->.(E)

T->.id

Goto(I0,E)=I1

I1:

A->E.

Goto(I0,T)=I2

I2:

E->T.E’

E’->.+TE’

E’->.

Goto(I0,()=I3

I3:

T->(.E)

E->.TE’

T-> .(E)

T->.id

Goto(I0,id)=I4

I4:

T->id.

Goto(I2,E’)=I5

I5:

E->TE’.

Goto(I2,+)=I6

I6:

E’->+.TE’

T->.(E)

T->.id

Canonical Parser

A->α.aβ,b is any In Goto(In,a)=Im into Action[n,a]

A->α.a In A->α to Action[n,a]

S’->S.,$ In Action[n,$]

Goto(In,A)=Im then add Goto(n,A)=m

A->α.aβ,b€(FIRST(β),a)

Check the following grammar is LR1 grammar or not.

S->CC

C->cC|d

Now we can find First() and Follow()

|  |  |  |
| --- | --- | --- |
|  | First | Follow |
| S | {c,d} | {$} |
| C | {c,d} | {$} |

Now we can find augmented grammar of above grammar,

S’->S

S->CC

C->cC

C->d

Now I0 item set is

I0:

S’->.S,$ A->α.aβ,b€(FIRST(β),a) here A=S’,α=€,a=S,β=€

S->.CC,$ A->α.aβ,b€(FIRST(β),a) here A=S,α=€,a=C,β=C

C->.cC,c/d

C->.d,c/d

Now we can find item set I1 i.e.Goto(I0,S)=I1

I1:

S’->S.,$

Now we can find item set I2 i.e.Goto(I0,C)=I2

I2:

S->C.C,$

C->.cC,$

C->.d, $

Now we can find item set I3 i.e.Goto(I0,c)=I3

I3:

C->c.C,c/d

C->.cC,c/d

C->.d, c/d

Now we can fin item set I4 i.e.Goto(I0,d)=I4

I4:

C->d.,c/d

Now we can find item set I5 i.e.Goto(I2,C)=I5

I5:

S->CC.,$

Now we can find item set I6 i.e.Goto(I2,c)=I6

I6:

C->c.C,$

C->.cC,$

C->.d, $

Now we can find item set I7 i.e.Goto(I2,d)=I7

I7:

C->d.,$

Now we can find item set I8 i.e.Goto(I3,C)=I8

I8:

C->cC.,c/d

Goto(I3,c)=I3

Goto(I3,d)=I4

Now we can find item set I9 i.e.Goto(I6,C)=I9

I9:

C->cC.,$

Goto(I6,c)=I6

Goto(I6,d)=I7

Now we can construct CLR parse table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Action | | | Goto | |
| c | d | $ | S | C |
| 0 | S3 | S4 |  | 1 | 2 |
| 1 |  |  | Accept |  |  |
| 2 | S6 | S7 |  |  | 5 |
| 3 | S3 | S4 |  |  | 8 |
| 4 |  |  | R3 |  |  |
| 5 |  |  | R1 |  |  |
| 6 | S6 | S7 |  |  | 9 |
| 7 |  |  | R3 |  |  |
| 8 |  |  | R2 |  |  |
| 9 |  |  | R2 |  |  |

Check whether following grammar is CLR grammar or not

S->B

A->A;E|E

B->bDAe

D->Dd;|€

E->B|a

|  |  |  |
| --- | --- | --- |
|  | FIRST | FOLLOW |
| S | {b} | {$} |
| A | {a,b} | {;,e} |
| B | {b} | {$,;,e} |
| D | {d,€} | {a,b,d} |
| E | {a,b} | {;,e} |

Now we can write augmented grammar

S’->S R0

S->B R1

A->A;E R2

A->E R3

B->bDAe R4

D->Dd; R5

D->€ R6

E->B R7

E->a R8

Now we can find out canonical item sets

I0:

S’->.S,$

S->.B,$

B->.bDAe,$

Now we can find item set I1,i.e.Goto(I0,S)=I1

I1:

S’->S.,$

Now we can find item set I2 i.e.Goto(I0,B)=I2

I2:

S->B.,$

Now we can find item set I3 i.e. Goto(I0,b)=I3

I3:

B->b.DAe,$

D->.Dd;,a/b

D->.€,a/b

Now we can find item set I4 i.e.Goto(I3,D)=I4

I4:

B->bD.Ae,$

A->.A;E,e

A->.E, e

E->.B,e

E->.a, e

B->.bDAe,e

D->D.d;,a/b

Now we can find item set I5 i.e.Goto(I4,A)=I5

I5:

B->bDA.e,$

A->A.;E,e

Now we can find item set I6 i.e.Goto(I4,E)=I6

I6:

A->E.,e

Now we can find item set I7 i.e.Goto(I4,B)=I7

I7:

E->B.,e

Now we can find item set I8 i.e.Goto(I4,a)=I8

I8:

E->a.,e

Now we can find item set I9 i.e.Goto(I4,b)=I9

I9:

B->b.DAe,e

D->.Dd;a/b/e

D->.€,a/b/e

Now we can find item set I10 i.e.Goto(I4,d)=I10

I10:

D->Dd.;,a/b

Now we can find item set I11 i.e.Goto(I5,e)=I11

I11:

B->bDAe. $

Now we can find item set I12 i.e.Goto(I5,;)=I12

I12:

A->A;.E e

E->.B e

E->.a e

Now we can find item set I13 i.e.Goto(I9,D)=I13

I13:

B->bD.Ae e

A->.A;E e

A->.E e

D->D.d;a/b/e

Now we can find item set I14 i.e.Goto(I10,;)=I14

I14:

D->Dd;. a/b

Now we can find item set I15 i.e.Goto(I12,E)=I15

I15:

A->A;E. e

Goto(I12,B)=I7

Goto(I12,a)=I8

Now we can find item set I16 i.e.Goto(I13,A)=I16

I16:

B->bDA.e e

A->A.;E e

Goto(I13,E)=I6

Now we can find item set I17 i.e.Goto(I13,d)=I17

I17:

D->Dd.; a/b/e

Now we can find item set I18 i.e.Goto(I16,e)=I18

I18:

B->bDAe. e

Goto(I16,;)=I12

Now we can find item set I19 i.e.Goto(17,;)=I19

I19:

D->Dd;. a/b/e

Now we can construct CLR parse table

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Action | | | | | | |  | Goto | | | |
| a | b | d | e | € | ; | $ | S | A | B | D | E |
| 0 |  | S3 |  |  |  |  |  | 1 |  | 2 |  |  |
| 1 |  |  |  |  |  |  | Accept |  |  |  |  |  |
| 2 |  |  |  |  |  |  | R1 |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  | 4 |  |
| 4 | S8 | S9 | S10 |  |  |  |  |  | 5 | 7 |  | 6 |
| 5 |  |  |  | S11 |  | S12 |  |  |  |  |  |  |
| 6 |  |  |  | R3 |  | R3 |  |  |  |  |  |  |
| 7 |  |  |  | R7 |  | R7 |  |  |  |  |  |  |
| 8 |  |  |  | R8 |  | R8 |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  | 13 |  |
| 10 |  |  |  |  |  | S14 |  |  |  |  |  |  |
| 11 |  |  |  | R4 |  | R4 | R4 |  |  |  |  |  |
| 12 | S8 |  |  |  |  |  |  |  |  | 7 |  | 15 |
| 13 |  |  | S17 |  |  |  |  |  | 16 |  |  | 6 |
| 14 | R5 | R5 | R5 |  |  |  |  |  |  |  |  |  |
| 15 |  |  |  |  |  |  |  |  |  |  |  |  |
| 16 |  |  |  | S18 |  | S12 |  |  |  |  |  |  |
| 17 |  |  |  |  |  | S19 |  |  |  |  |  |  |
| 18 |  |  |  | R4 |  | R4 | R4 |  |  |  |  |  |
| 19 | R5 | R5 | R5 |  |  |  |  |  |  |  |  |  |

Check whether following grammar is CLR grammar or not.

S->E=E|a

E->E+T|T

T->T\*a|a

Now we can find FIRST() and FOLLOW()

|  |  |  |
| --- | --- | --- |
|  | FIRST | FOLLOW |
| S | {a} | {$} |
| E | {a} | {=,+} |
| T | {a} | {=,+,\*} |

The augmented grammar is

S’->S

S->E=E

S->a

E->E+T

E->T

T->T\*a

T->a

Now we can find canonical item set I0

I0:

S’->.S $

S->.E=E $

S->.a $

E->.E+T =

E->.T =

T->T\*a =

T->a =

Now we can find item set I1 i.e.Goto(I0,S)=I1

I1:

S’->S. $

Now we can find item set I2 i.e.Goto(I0,E)=I2

I2:

S->E.=E $

E->E.+T =

Now we can find item set I3 i.e.Goto(I0,a)=I3

I3:

S->a. $

T->a. =

Now we can find item set I4 i.e.Goto(I0,T)=I4

I4:

E->T. =

T->T.\*a =

Now we can find item set I5 i.e.Goto(I2,=)=I5

I5:

S->E=.E $

E->.E+T $

E->.T $

T->.T\*a $

T->.a $

Now we can find item set I6 i.e.Goto(I2,+)=I6

I6:

E->E+.T =

T->.T\*a =

T->.a =

Now we can find item set I7 i.e.Goto(I4,\*)=I7

I7:

T->T\*.a =

Now we can find item set I8 i.e.Goto(I5,E)=I8

I8:

S->E=E. $

E->E.+T $

Now we can find item set I9 i.e.Goto(I5,T)=I9

I9:

E->T. $

T->T.\*a $

Now we can find item set I10 i.e.Goto(I5,a)=I10

I10:

T->a. $

Now we can find item set I11 i.e.Goto(I6,T)=I11

I11:

E->E+T. =

T->T.\*a =

Goto(I6,a)=I3

Now we can find item set I12 i.e.Goto(I7,a)=I12

I12:

T->T\*a. =

Now we can find item set I13 i.e.Goto(I8,+)=I13

I13:

E->E+.T $

T->.T\*a $

T->.a $

Now we can find item set I14 i.e.Goto(I9,\*)=I14

I14:

T->T\*.a $

Goto(I11,\*)=I7

Now we can find item set I15 i.e.Goto(I13,T)=I15

I15:

E->E+T. $

T->T.\*a $

Goto(I13,a)=I10

Now we can find item set I16 i.e.Goto(I14,a)=I16

I16:

T->T\*a. $

Goto(I15,\*)=I14

Now we can combine similar core part item together and find find canonical item set for LALR

I0:

S’->.S $

S->.E=E $

S->.a $

E->.E+T =

E->.T =

T->T\*a =

T->a =

I1:

S’->S. $

I28:

S->E.=E $

E->E.+T = /$

I3:

S->a. $

T->a. =

I49:

E->T. = /$

T->T.\*a =/$

I15:

E->E+T. $

T->T.\*a $

I613:

E->E+.T = /$

T->.T\*a =/$

T->.a =/$

I714:

T->T\*.a =/$

I10:

T->a. $

I1115:

E->E+T. =/$

T->T.\*a =/$

I1216:

T->T\*a. =/$

Evaluation order for semantic rules

Dependancy Graph

Syntax Tree

Input Text

Type\_error()

Void

Typedef int \*intr

Array(I,T)

TXP

Struct structname

{};

Float \*xyz

Int fun()

|  |
| --- |
| S->D;E |
| D->D;D| T LIST |
| LIST->\*L1 |
| LIST->id |
| T->char |
| T->int |
| T->float |
| LIST->\*L1 |
| LIST->L1[num] |
| E->literal|num|id|E mod E|E[E]|\*E|E op E |
|  |
| Production Rule | Type Expression |
| S->D;E | This means all declarations before expression |
| S->T LIST | LIST.type=T.type |
| LIST->id | {addtype(id.entry,LIST.type)} |
| T->char | {T.type:=char} |
| T->int | {T.type:int} |
| T->float | {T.type=float} |
| LIST->\*L1 | {L1.type:=pointer(LIST.type)} |
| LIST->L1[num] | {L1.type:=array(0….num,val-1,LIST.type)} |
|  |  |

E->literal{E.type:=char}

e->num{E.type:=int}

ex.

S1 S2 Equivalence Reason

Char char s1 is equivalent s2 simillar basic type

Pointer(char) pointer(char) s1 is equivalent s2 simmilar pointer type

typedef struct Node

{

int x;

}node;

Node \*first,\*second;

Struct Node \*last1,\*last2;

-a\*b+-a\*b

a-\*b+a-\*b

a-b\*+a-\*b

a-b\*a-+\*b

a-b\*a-+b\*

a:=b op c

a=b+c+d

t1=b+c

t2=t1+d

t3=t2

a=t3

x:=y op z

x:=opy

x:=y

goto L

if x relop y

goto L

param X1,

param x2,

….

Return y

X=y[i]

X=&y

X=x\*y

\*x=y

-a\*b+-a\*b

The three address statement is

t1:= -a

t2:=t1\*b

t3=-a

t4=t3\*b

t5=t2+t4

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Arg1 | Arg2 | Result |
| - | a |  | T1 |
| \* | T1 | b | T2 |
| - | a |  | T3 |
| \* | T3 | b | T4 |
| + | T2 | T4 | T5 |

**Difference between Top Down parser and Bottom Up parser-**

**Top Down Parser Bottom Up Parser**

1)This parser uses derivation process. 1)This parser uses reduction process.

2)It does not accept ambiguous grammar. 2)It accepts ambiguous grammar.

3)It will not accept left recursion. 3)It will accept left recursion grammar.

4)Backtracking can be avoided using left 4)This parser can not avoid backtracking.

Factoring grammar.

5)Precise error indication is not possible. 5)Precise error indication is possible.

6)There are following types of parser 6)Types of bottom up parser

a)Backtracking a)Shift reduce parser

b)Predictive b)Operator precedence parser

c)LR parser

7)It uses LL1 grammar for parsing. 7)It uses SLR1 grammar, operator

grammar, canonical grammar etc.

8)It uses LL1 parse table for storing grammar. 8)It uses SLR1,canonical,LALR parse table

for storing grammar.

9)Data structure and designing process of LL1 9)Data structure and designing process of

parse table is very simple. But it is not LR parse table is very complicated. But it

intelligent table. It only supplies the production is intelligent table, it not only supply prod

rule. -uction rule but also tell where to go and

which operation should be performed on

that movement.

10)It is slower than bottom up. 10)It is faster than top down.

11)Logic of program and parse table is very 11)Logic of programs are very simple but

simple. construction of parse table is tedious job.

**Difference between LL Parser and LR Parser-**

**LL Parser LR Parser**

i)LL parsers begin at the start symbol and try i)LR parser begins at the target string and try

to apply productions to arrive at the target to arrive back at the start symbol.

string.

ii)It is based on the principle top-down ii) It is based on the principle bottom-up parsing. parsing.

iii)It uses leftmost derivation. iii)It uses rightmost derivation.

iv)This parser based on two actions. iv)In LR parser there are two actions.

a)predict b)match a)shift b)reduce

v)LL parser are less powerful than LR parser v)LR parsers are powerful they accept large se

because it accepts small set of grammars. -t of grammars.

vi)They are less intelligent. vi)Parser table of LR parser intelligent table be

-cause it tells which operation or action shoul

-d perform on which condition. This table is

divided into two parts.

a)Action part b)Goto part

**Items-**

-To create a parsing table for grammar we must introduce a special symbol, which indicates the current position for which the parser has already read symbols on the input and what to expect next.

Ex.E→E·+B

This shows that E has already been processed and the parser is looking for at symbol next.

-Each of these above rules is called item.

-There is an item for each position that dot symbol can take along the right hand side of the rule.

**Item sets-**

A parser may not know which grammar rule to use in advance, when creating our table we must use set of items to consider all possibilities.

Ex.S→·E

-Every SLR can be converted into CLR but not every CLR can be converted into SLR.

-Speed of parsing is slow.

**Canonical LR Parse table-**

-The canonical set of items is the parsing technique in which a lookahead symbol is generated while constructing set of items. Hence, the collection of set of items is referred as LR(1). The value 1 in the bracket indicates that there is one lookahead symbol in the set of items.

-A canonical LR parser or LR(1) parser is an LR(k) parser for k=1 that is with a single lookahead terminal.

-The special attribute of this parser is that all LR(k) parsers with k>1 can be transformed into a LR(1) parser.

-It can handle all deterministic CFG.

-In past this LR(k) parser has been avoided because of it’s huge memory requirements in favor of less powerful alternatives such as LALR and LL(1).

-Both FIRST and FOLLOW functions can be used.

**Procedure of finding out CLR parse table-**

Construct CLR parse table using following rules.

1)Divide parse table into two parts that is Action part and GOTO part.

2)Assume there are total n items that are I0, I1,………… ,In.

There are total n+1 rows of Action table as well as GOTO table.

3)All terminals are column headers of the action table and all non terminals are the column headers of GOTO table.

4)If item A→α·aβ,b is any In, and GOTO (InA)=Im then add shift m into Action[n,a].

5)If item A→α·a is any In, and A is not starting non terminal then add reduce A→α to Action[n,a] for all a mentioned as FIRST of that non terminal that is Action[n,a]=rm where mth production rule is A->α used for reduction.

A→α·aβ,b€(First(β),a)

6)If S1→S·$is many. In then add accept in Action[n,$]that is Action[n,$]=accept.

7)If there is GOTO function like GOTO(In,A)=Im then add GOTO[n,A]=m.

8)All entries which are not defined, make them as error.

If all cell of the parse table contain only one entry then grammar is called CLR grammar otherwise it is not CLR grammar.

**1)Check following grammar is LR(1) grammar or not**

**S→CC**

**C→cC|d**

**Answer**

The augmented grammar of above grammar as,

R0  S1→S

R1 S→CC

R2 C→cC

R3 C→d

Now, we find FIRST of all non-terminals.

FIRST(S1)=FIRST(S)=FIRST=(C)={c,d}

Now, we find out LR(1) items.

Find out I0 items

I0:S1→**·**S,$ →Now **·** is just before S ∴Closure is evaluated

S→·CC,$ →S productions are added with FIRST($)

Now · just before C ∴Closure is operated on C. All productions are added with FIRST{C,$}

C→·cC,c|d

C→·d,c|d

Now, find out I1 item using GOTO function

∴GOTO(I0,S)=I1

I1:S1→S·$

GOTO function is operated on every ·S of I0 in I1.

·Operator is shifted after E.

Now, find out I2 item using GOTO function i.e. GOTO(I0,C)=I2

I2:S→C·C$

C→·cC,$

C→·d,$

GOTO function is operated on every ·C of I0 ,in I2 · operator is just shifted after C and closure is evaluated.

Now, find out I3 item using GOTO function.

I3:S→C·C,c|d

C→·cC,c|d

C→·d

GOTO function is operated on every ·C of I0 ,in I3 · operator is just shifted after C and closure is evaluated.

Now, find out I4 item using GOTO function i.e. GOTO(I0,d)= I4

I4:C →d· c|d →GOTO function is operated on every ·S of I0,I4. ·operator is just shifted after d.

Now, find out I5 item using GOTO function i.e. GOTO(I2,C)=I5

I5:S→CC.$

→ GOTO function is operated on every ·S of I2 ,in I3 · operator is just shifted after C.

Now, find out I6 item using GOTO function i.e. GOTO(I2,c)=I6

I6:C→c·C,$

C→·cC,$

C→·d,$

→ GOTO function is operated on every ·S of I2 ,in I6 · operator is just shifted after C.

Now, find out I7 item using GOTO function i.e. GOTO(I2,d)=I7

I7:C→d·$

→ GOTO function is operated on every ·S of I2 ,in I7 · operator is just shifted after d.

Now, find out I8 item using GOTO function i.e. GOTO(I3,C)=I8

I8:C→cC·,c|d

→ GOTO function is operated on every ·S of I5 ,in I8 · operator is just shifted after C.

Now, find out GOTO(I6,C)=I3

Similarly GOTO(I6,d)=I4

Now, find out I9 item using GOTO(I6,C)=I9

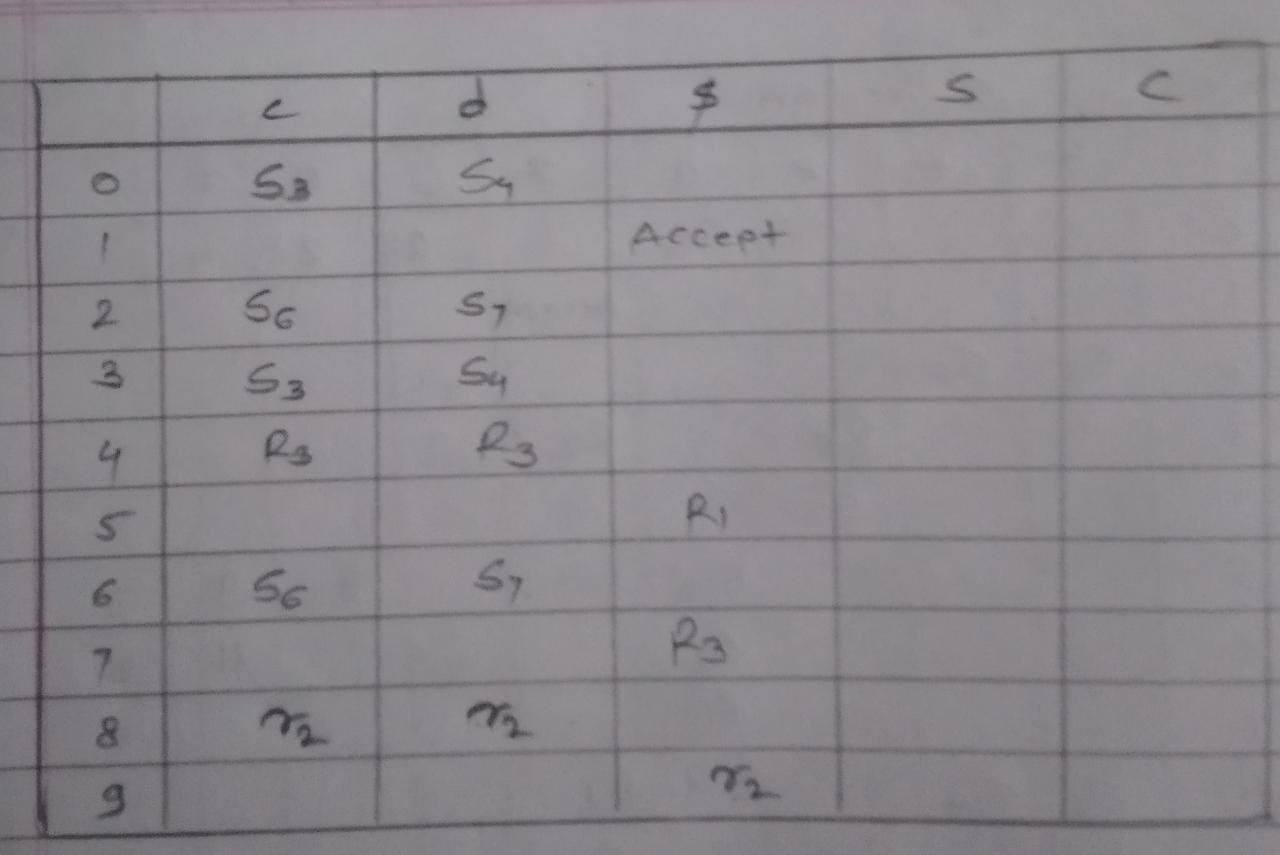
I9:C→cC·,$

GOTO function is operated on every ·S of I2 in I9 ·operator is just shifted after C.

Now, find out GOTO(I6,C)=I6

Similarly, GOTO(I6,d)=I7

No more items can be generates.



**2)Check following grammar is LR(1) grammar or not?**

**S→B**

**A→A;E|E**

**B→bDAe**

**D→Dd;|∊**

**E→B|a**

**Answer:**

Find out FIRST of all non-terminals

FIRST(S)=FIRST(B)={b}

FIRST(A)=FIRST(E)={a,b}

FIRST(D)={∊,d,;}

The augmented grammar for above grammar is

R0 S1→S

R1 S→B

R2 A→A;E

R3 A→E

R4 B→bDAe

R5 D→Dd;

R6 D→∊

R7 E→B

R8 E→a

Now, find out LR(1) items.

Find out I0 items.

I0:S1→·S,$

S→·B,$

B→·bDAe,$

GOTO(Io,S)=I1

I1:S1→S·,$

GOTO(I0,B)=I2

I2:S→B·,$

GOTO(I0,b)=I3

I3:Bb.DAe,$

D→·Dd;,a|b|d

D→·,a|b|d

GOTO(I3,D)=I4

I4:B→bD.Ae,$

A→·A;E,

R0S’→·S,$

R1S→B

R2A→A;E

R3A→E

R4B→bDAe

R5D→Dd;$

R6D→∊

R7E→B

R8E→a

FIRST(S)={b}

FIRST(A)={a,b}

FIRST(B)={b}

FIRST(D)={d,∊}

FIRST(E)={a,b}

I0:S’→·S,$

S→·B,$

B→·bDAe,$

GOTO(I0,S)=I1

I1:S’→S·,$

GOTO(I0,B)=I2

I2:S→B·,$

GOTO(I0,b)=I3

I3:B→b·DAe,$

D→·Dd;,a|b

D→∊,

3)Check following grammar is CLR(1) grammar or not?

S→E=E|a

E→E+T|T

T→T\*a|a

→Now, we find first of all non-terminals FIRST(S)=FIRST(E)=FIRST(T)={a}

Now, we compute LR(1)items I0.Now, we find LR(0) items.

Find out I0 items.

I0:S’→·S,$

S→·E=E,$

S→·a,$

E→·E+T,+|=

E→·T,+|=

T→·T\*a,+|=|\*

T→·a,+|=|\*

GOTO(I0,S)=I1

I1:S’→S·,$

GOTO(I0,E)=I2

I2:S→E·=E,$

E→E·+T,+|=

GOTO(I0,a)=I3

I3:S→a·,$

T→a\*,+|=|\*

GOTO(I0,T)=I4

I4:E→T·,+|=

T→T·\*a,+|=|\*

GOTO(I2,=)=I5

I5:S→E=·E,$

E→·E+T,+|$

E→·T,+|$

T→·T\*a,$|+|\*

T→·a,$|+|\*

GOTO(I2,+)=I6

I6:E→E+·T,+|=

T→·T\*a,+|=|\*

T→·a,+|=|\*

GOTO(I4,\*)=I7

I7:T→T\*·a,+|=|\*

GOTO(I5,E)=I8

I8:S→E=E·$

E→E·+T,+|$

GOTO(I5,T)=I9

I9:E→T·,+|$

T→T·\*a,$|+|\*

GOTO(I5,a)=I10

I10:T→a·,$|+|\*

GOTO(I6,T)=I11

I11:E→E+T·,+|=

T→T·\*a,+|=|\*

GOTO(I6,a)=I12

I12:T→a·,+|=|\*

GOTO(I7,a)=I13

I13:T→T\*a·,+|=|\*

GOTO(I8,+)=I14

I14:E→E+·T,+|$

T→·T\*a,+|$|\*

T→·a,+|$|\*

GOTO(I9,\*)=I15

I15:T→T\*·a,+|$|\*

GOTO(I14,T)=I16

I16:E→E+T·,+|$

T→T·\*a,+|$|\*

GOTO(I14,a)=I17

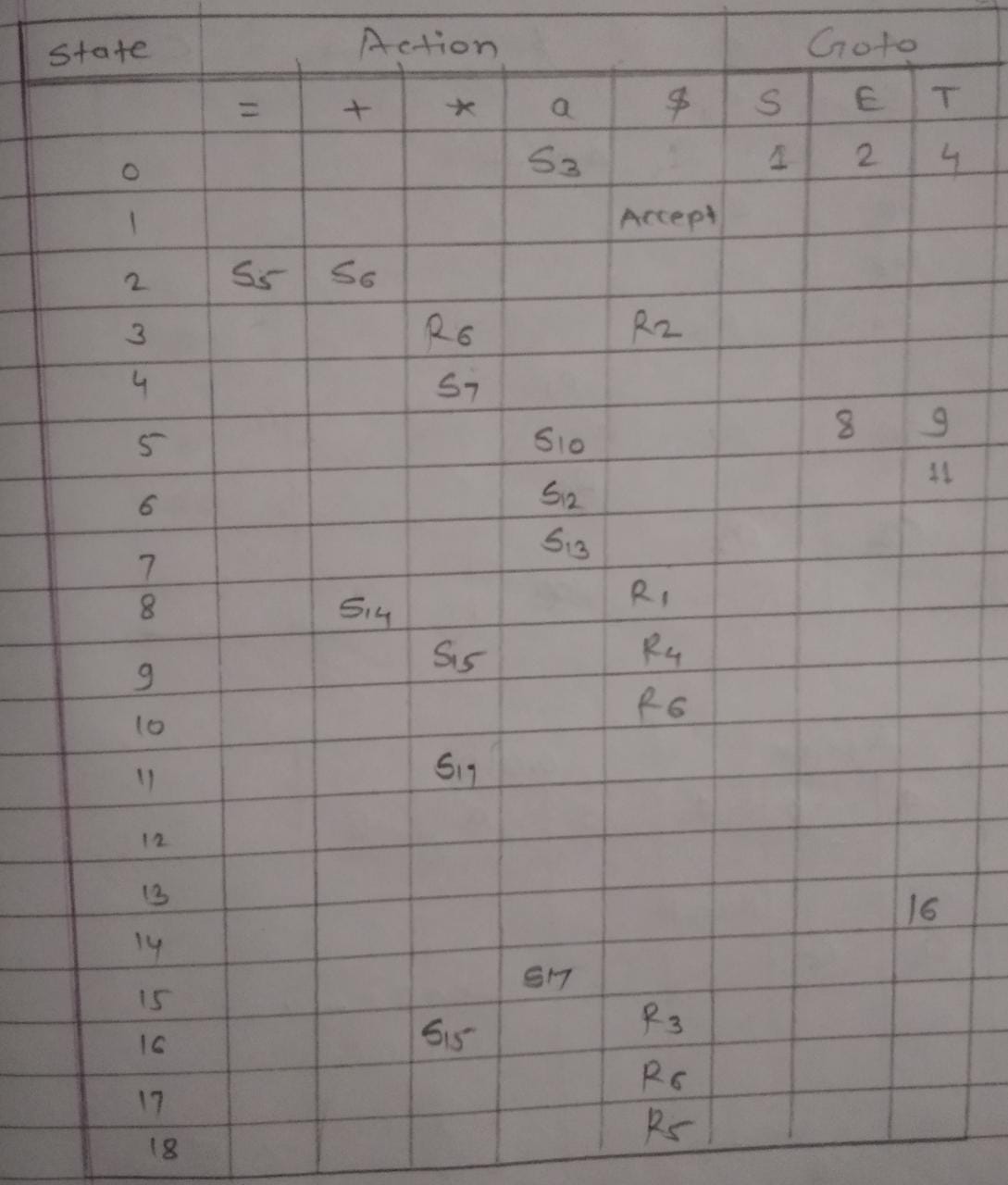
I17:T→a·,+|$|\*

GOTO(I16,\*)=I15

No, more items can be generated

GOTO(I15,a)=I18

T→T\*a·,+,$|\*



**Closure Function-**

The closure function can be computed as, for each item A→α·Xβ,a and rule X→ɣ

and b∊FIRST(β,a)

such that, X→·ɣ and b is not in I then add X→·ɣ, b to I

LR(1) item-

An LR(1) item is a two component element of the form [A→α·β,u]

Where, first component is marked production A→α·β is called core of the item and u is a lock ahead character that belongs to set V.

**Construct canonical LR(1) item-**

**1)S→AaAb**

**S→BbBa**

**A→∊**

**B→∊**

**Answer:**

FIRST(S)={a,b}

FIRST(A)=FIRST(B)=∊

Now, we find out LR(1) item.

I0:S’→·S,$

S→·AaAb,$

A→·,a

B→·,b

GOTO(I0,S)=I1

I1:S’→S·,$

GOTO(I0,A)=I2

I2:S→A·aAb,$

GOTO(I0,B)=I3

I3:S→B·bBa,$

GOTO(I2,a)=I4

I4:S→Aa·Ab,$

A→·,b

GOTO(I2,b)=I5

I5:S→Bb·Ba,$

B→·,a

GOTO(I4,A)=I6

I6:S→AaA·b,$

GOTO(I5,B)=I7

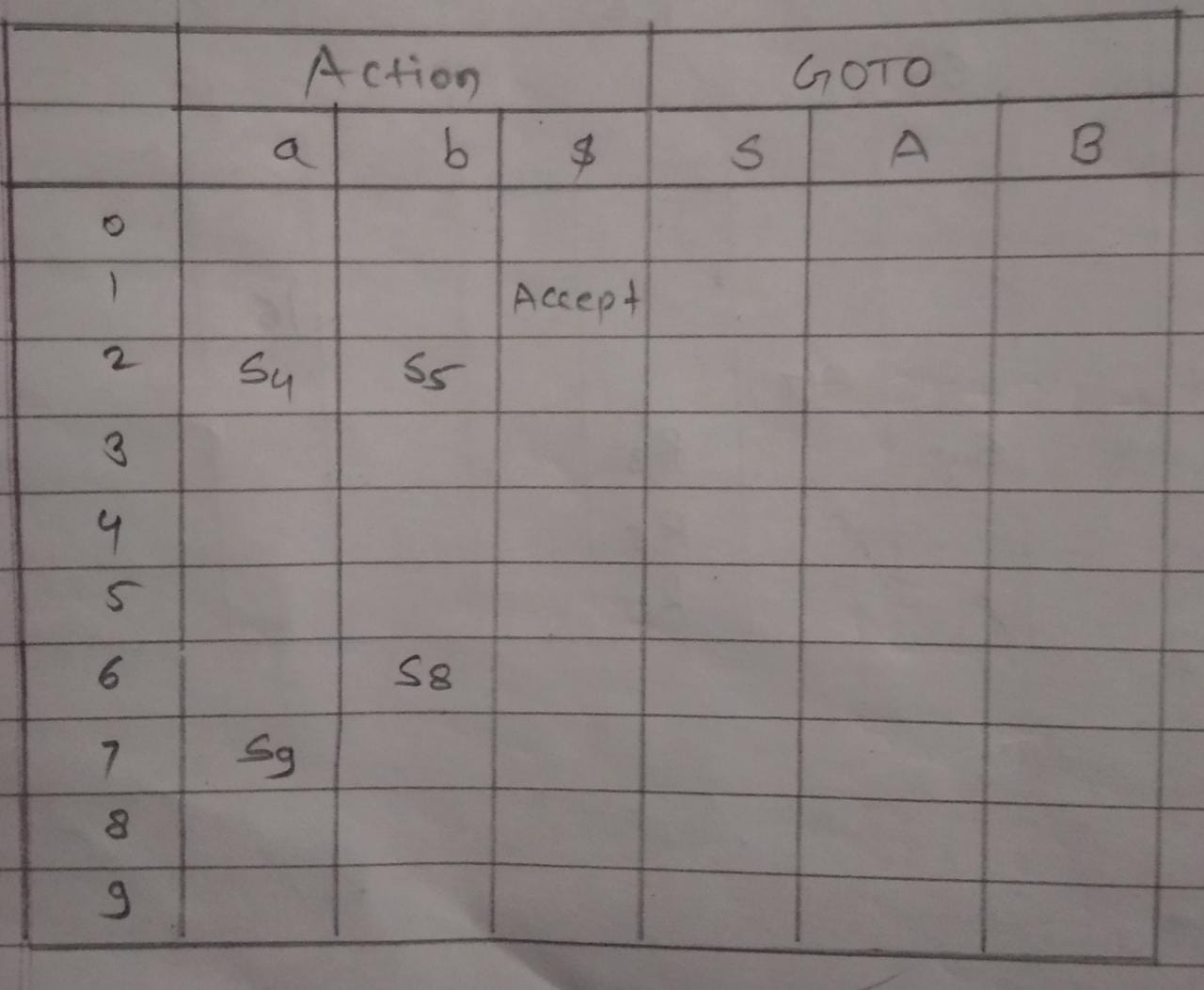
I7:S→BbB·a,$

GOTO(I6,b)=I8

I8:S→AaAb·,$

GOTO(I7,a)=I9

I9:S→BbBa·,$



FIRST(S)={c,d}, FIRST(C)={c,d}

**LALR Parser-**

-Look ahead LR parser is called as LALR.

-Table obtained by it are smaller than the canonical LR tables.

-LALR tables are constructed by replacing all states having the same core with union of CLR parse table.

-Generally, grammar of any high level statement is LALR grammar as well as SLR grammar.

-In fact states of SLR and LALR parsing are always same.

-Most of the programming language uses LALR.

-Procedure for constructing LALR tables.

1)Find out augmented grammar of that grammar.

2)Find out LR(1) items like I0,I1,I2,I3 using closure function and GOTO function.

3)Construct CLR parse table using rules mentioned in above section. Find out states or items which has same core.

4)Combine them.

5)Check any conflict is there or not? If there is no conflict them grammar is LALR grammar, otherwise not LALR.

**1)Construct set of LR(1) items for LALR parser as well as construct the parsing table for LALR parser for following grammar.**

**S→CC**

**C→aC**

**C→d**

**Answer:**

Now, first we construct LR(1) items.

The augmented grammar is

R0 S’→S

R1 S→CC

R2 C→aC

R3 C→d

Now, we find out LR(1) item.

I0:S’→·S,$

A=S’, α=∊, X=S, β=∊, a=$

∴Then we add S productions

S→·CC, b∊ FIRST(β,a)

b∊ FIRST(∊,$)

b∊ FIRST($)

b={$}

S→·CC,$

A=S,α=∊,X=C,β=C,a=$

We add C productions.

C→·aC, b∊ FIRST(β,a)

b∊ FIRST(C,$)

b∊ FIRST(a,d)

∴[C→a·C,a/d]

C→·d, b∊ FIRST(β,a)

b∊ FIRST(C,$)

b∊ FIRST(a/d)

C→·d,a/d

∴I0 items are

I0:S’→·S,$

S→·CC,$

C→·aC,a/d

C→·d,a/d

GOTO(I0,S)=I1

I1:S’→S·,$

GOTO(I0,C)=I2

I2:S→C·C$

A→α·Xβ,a

∴A=C,α=C,X=C,β=∊,a=$

We add C productions

C→·aC, b∊ FIRST(β,a)

b∊ FIRST(∊,$)

b∊ FIRST($)

b={$}

C→·d, b∊ FIRST(β,a)

b={$}

∴ I2 items are

I2:S→C·C,$

C→·aC,$

C→·d,$

GOTO(I0,a)=I3

I3:C→a·C,a/d

A→α·Xβ,a

∴A=C,α=a,X=C,β=∊,a=a/d

We add C productions

C→·aC, b∊ FIRST(β,a)

b∊ FIRST(a,d)

b∊ FIRST(a,d)

C→·aC,a/d

C→·d, b∊ FIRST(β,a)

b∊ FIRST(a,d)

b={a,d}

C→·d,a/d

I3 items are

I3:C→a·C,a/d

C→·aC,a/d

C→·d,a/d

GOTO(I0,d)=I4

I4:C→d·,a/d

GOTO(I2,C)=I5

I5:S→CC·$

GOTO(I2,a)=I6

I6:C→a·C,$

A→α·Xβ,a

∴A=C,α=a,X=C,β=∊,a=$

We add C productions

C→·aC, b∊ FIRST(β,a)

b∊ FIRST(∊,$)

b=$

C→·aC,$

C→·d, b∊ FIRST(β,a)

b=$

C→·d,$

I6 items are

I6:C→a·C,$

C→·aC,$

C→·d,$

GOTO(I2,d)=I7

I7:C→d·,$

GOTO(I3,C)=I8

I8:C→aC·,a/d

GOTO(I6,C)=I9

I9:C→aC·,$

Now, we will merge states 3,6 then 4,7 and 8,9

Then the LR(1) item sets are

I0:S’→·S,$

S→·CC,$

C→·aC,a/d

C→·d,a/d

I1:GOTO(I0,S)

S’→S·,$

I2:GOTO(I0,C)

S→C·C,$

C→·aC,$

C→·d,$

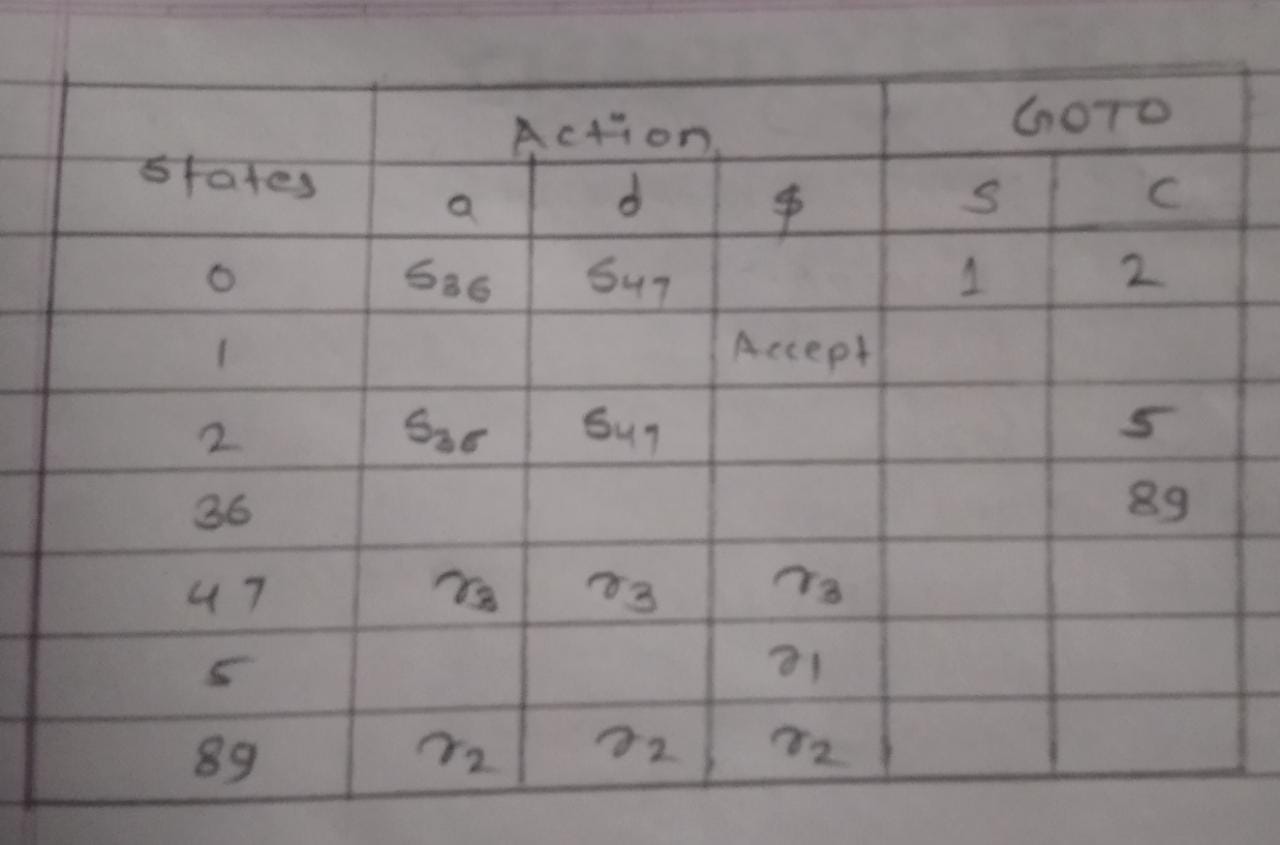
I36:GOTO(I0,a)

C→a·C,a/d|$

C→·aC,a/d|$

C→·d,a/d|$

The LALR parse table is



**Q. Check whether the following grammar is CLR(1).**

**S→aAb|bSA**

**A→bB|C**

**B→CA|d**

**Answer**

In above grammar C is useless symbol then grammar after eliminating C is

S→aAb|bSA

A→bB

B→d

Now, we make the above grammar as augmented

R0 S’→S

R1 S→aAb

R2 S→bSA

R3 A→bB

R4 B→d

Now, we find LR(1) items

I0:S’→·S,$

A→α·Xβ,a

Where A=S’, α=∊,X=S,β=∊,a=$

∴we add productions of S

S→·aAb, b∊ FIRST(β,a)

b∊ FIRST(∊,$)

b∊ FIRST($)

b={$}

S→·aAb,$

S→·bSA, b∊ FIRST(β,a)

b∊ FIRST(∊,$)

b∊ FIRST($)

b={$}

S→·bSA,$

∴I6 items are,

I0:S’→·S,$

S→·aAb,$

C→·bSA,$

FIRST(S)={a,b}

FIRST(A)={b}

FIRST(B)=d

GOTO(I0,S)=I1

I1:S’→S·,$

GOTO(I0,a)=I2

I0:S→a·Ab,$

A→α·Xβ,a

Where A=S, α=a, X=A,β=b,a=$

∴we add productions of A

A→·bB, b∊ FIRST(β,a)

b∊ FIRST(b,$)

b∊ FIRST(b)

A→·bB,b

∴I2 items are

I2:S→a·Ab,$

A→·bB,b

GOTO(I0,b)=I3

I3:S→b·SA,$

A→α·Xβ,a

Where A=S, α=b, X=S,β=A,a=$

∴we add X productions

S→·aAb, b∊ FIRST(β,a)

b∊ FIRST(A,$)

b∊ FIRST(b)

S→·aAb,b

S→·bSA, , b∊ FIRST(β,a)

b∊ FIRST(A,$)

b∊ FIRST(b)

S→·bSA,b

∴I3 items are

I3:S→b·SA,$

S→·aAb,b

S→·bSA,b

GOTO(I2,A)=I4

I4:S→aA·b,$

GOTO(I2,b)=I5

I5:A→b·B,b

A→α·Xβ,a

∴we add productions of B

B→·d, b∊ FIRST(β,a)

b∊ FIRST(∊,b)

B→·d,b

I5 items are

I5:A→b·B,b

B→·d,b

GOTO(I3,S)=I6

I6:S→bS·A,$

A→α·Xβ,a

Where A=S, X=A,β=∊,a=$

∴we add A productions

A→·bB, b∊ FIRST(β,a)

b∊ FIRST(∊,$)

b∊ FIRST($)

S→·bB,$

I6:S→bS·A,$

A→·bB,$

GOTO(I3,a)=I2

GOTO(I3,b)=I3

GOTO(I4,b)=I7

I7:S→aAb·,$

GOTO(I5,B)=I8

I8:A→bB·,b

GOTO(I5,d)=I9

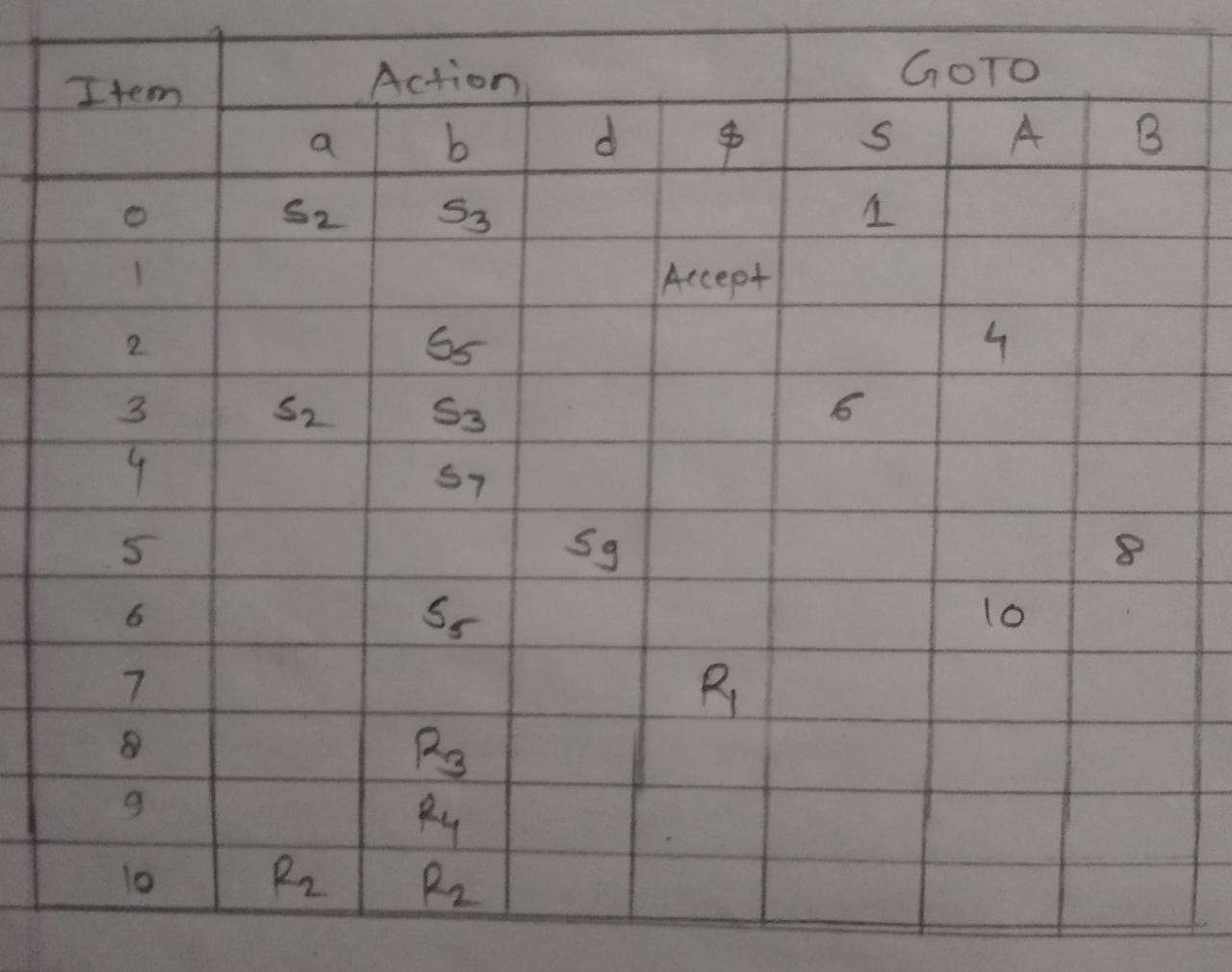
I9:B→d·

GOTO(I6,A)=I10

I10:S→bSA·,$

GOTO(I6,b)=I5

No, more item can be generated CLR table.



In above parse table there is not multiple entry in any cell, so above grammar is CLR grammar.

GOTO(I15,E)=I7

GOTO(I15,d)=I18

D→Dd·;a|b|e

GOTO(I17,e)=I19

B→bDAe·,e

GOTO(I17,;)=I10

The canonical item sets for

I0:S’→·S,$

S→·B,$

B→·bDAe,$

GOTO(I0,S)=I1

I1:S’→·S,$

GOTO(I0,B)=I2

I2:S→B·,$

GOTO(I0,b)=I3

I3:B→b·DAe,$

D→·Dd;a/b

D→·∊,a/b

GOTO(I3,D)=I4

I4:B→bD·Ae,$

A→·A;E,e

A→·E,e

D→D·d;a/b

GOTO(I3,∊)=I5

I5:D→∊·,a/b

GOTO(I4,A)=I6

I6:B→bDA·e,$

A→A·;E,e

GOTO(I4,E)=I7

I7:A→E·e

GOTO(I4,d)=I8

I8:D→Dd·;,a/b

GOTO(I6,e)=I9

I9:B→bDAe·,$

GOTO(I6,;)=I10

I10:A→A;·E,e

E→·B,e

B→·bDAe,e

E→·a,e

GOTO(I10,E)=I11

I11:A→A;E·,e

GOTO(I0,B)=I12

I12:E→B·,e

GOTO(I10,b)=I13

I13:B→b·DAe,e

D→·Dd;,a/b/|e

D→·∊,a/b|e

GOTO(I10,a)=I14

I14:E→a·,e

GOTO(I13,D)=I15

I15:B→bD·Ae,e

A→·A;E,e

D→D·d;,,a/b|e

GOTO(I13,∊)=I16

I16:D→·∊,a/b|∊

GOTO(I15,A)=I17

I17:B→bDA·e,e

A→A·;E,e

**\*Construct LR(1) parser**

S→Aa|dAb|dCa|Cb, A→C

→The augmented grammar is

R0 S’→S

R1 S→Aa

R2 S→dAb

R3 S→dCa

R4 S→Cb

R5 A→C

I0:S’→·S,$

S→·Aa,$

S→·dAb,$

S→·dCa,$

S→·Cb,$

A→·C,a

GOTO(I0,S)=I1

I1:S’→S·,$

GOTO(I0,A)=I2

I2:S→A·a,$

GOTO(I0,d)=I3

I3:S→d·Ab,$

A→·C,b

S→d·Ca,$

GOTO(I0,C)=I4

I4:S→C·b,$

A→C·,$

GOTO(I2,a)=I5

I5:S→Aa·,$

GOTO(I3,A)=I6

I6:S→dA·b,$

GOTO(I3,C)=I7

I7:A→C·,b

S→dC·a,$

GOTO(I4,b)=I8

I8:S→Cb·,$

GOTO(I6,b)=I9

I9:S→dAb·,$

GOTO(I7,a)=I10

I10:S→dCa·,$

FIRST(S)={C,d}

FIRST(A)={C}

