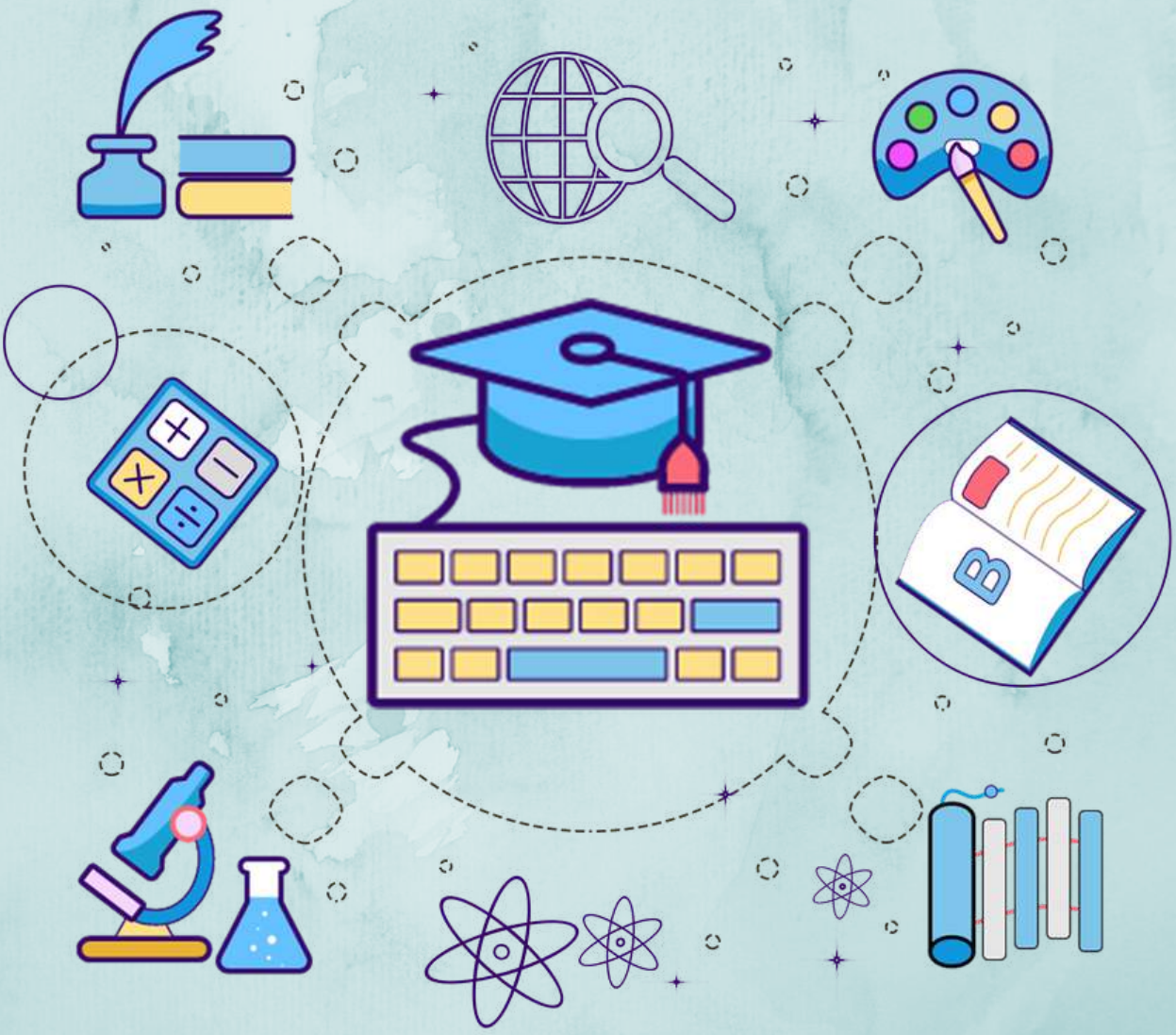
APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

# KeralaNotes



SYLLABUS | STUDY MATERIALS | TEXTBOOK

PDF | SOLVED QUESTION PAPERS

www.keralanotes.com

KeralaNotes

# KTU STUDY MATERIALS

# COMPREHENSIVE COURSE WORK

Related Link :

- KTU S6 NOTES 2019 SCHEME
- KTU S6 SYLLABUS COMPUTER SCIENCE
- KTU PREVIOUS QUESTION BANK S6 SOLVED
- KTU TEXTBOOKS S6 B.TECH PDF DOWNLOAD
- KTU S6 NOTES | SYLLABUS | QBANK | TEXTBOOKS DOWNLOAD

# CHAPTER – 1

# OVER VIEW

## Concepts & Definitions

- **Language Processor :**Computers, Interpreter, Assembler, normally an integrated product development environment
- **Phase of a compiler :** Using the divide & conquer strategy we break up the compiler into phases. Each phase transforms the source program into an intermediate form suitable for the next phase.

| Phase | Intermediate form | Next Phase |
|---|---|---|
| 1. Develop Program | Source program | Scanner |
| 2. Scanner / Lexical Analyzer | Tokens | Syntax Analyzer |
| 3. Syntax Analyzer | Syntax Tree | Semantic Analysis |
| 4. Semantic Analyzer | Intermediate Forms (ex: quadruples) | Code Optimization |
| 5. Code Optimization | Object Code | Linking, Loading & Execution |

- **Machine Language :** First generation language programming done in binary.
- **Assembly Language:** Second Generation languages names or mnemonics given to numbers
- **High level Language:** Close to natural languages
- **Models in LP:** Automata, grammar, regular expressions, trees etc;
- **Code optimization:** undecidable in theory. In practice code as good as that written by an expert human programmer is produced.
- **Block structure:** Nesting & Blocks & naming conversions with scope rules
- **Parameter Parses:** Actual formal parameter correspondences
- **Aliasing :** Two or more names for the same object.
- **Grammar:** Define Hierarchical Structures uses terminal and meta symbols called non-terminals has a finite set of productions or rewriting rules.
- **Attributes :** Each rules of a grammar has an attribute



For More Study Materials : www.keralanotes.com

- **Lexical Analyzer:** Translates the source progam to tokes isolates the lexemes in the source program.
- **Parsing:** Syntax or form is checked by the rules of a grammar, Normally a derivation tree is constructed.
- **Predictive parser :** The flow of parsing is unambiguous by looking at one symbol ahead.
- **Synthesized Attributes:** Deal with semantic analysis with bottom up parsing.
- **Intermediate code**: Abstract parsing Syntax tree or three address codes can be used
- **Symbol table**: The heart of the compiler
- **Concept** : A complier can be considered to be a file conversion program
- **Concept**: The target code may be code in a HLL like on software conversion projects.
- **Boot strapping**: Compiler that generates newer versions of themselves.
- A good compiler generates correct, truthful code.
- A good compiler conforms to the source language standard
- A good compiler can handle program of any size
- A good compiler does not have quadratic or worse algorithms
- A compiler that can run on different platforms is called portable
- A compiler that produces target code for different platforms is re-targetable
- Target code optimizations are dangerous but useful
- Lot of compiler work is automatic generations from formalisms
- A context free grammar recovers the structure a linear programs hides.

## GRAMMARS

- **Grammars** : Finite set of syntax rules with which we can define infinite number of sentences :

  **Exc**: S $\rightarrow$a S / Sa / a

  Three simple rules define infinite strings like a,aa,aaa,……

- Grammars can be classified into Ambiguous and Unambiguous grammars based on the number of parse trees that exist for a given input string.



For More Study Materials : www.keralanotes.com

- **Ambiguous grammar:** For a given input string if there exist more than one parse tree, then the grammar is called ambiguous grammar.

  **Exc**: G: S $\rightarrow$a S / Sa / a is ambiguous for input string w= "aaa"
- There is no algorithm that converts an ambiguous grammar to unambiguous grammar.
- Grammars can be classified into two types.
  1. Left recursive
  2. Right recursive – based on the way the productions are defined
- **Left Recursive Grammar:** In a grammar, in any rule, if L.H.S non-terminal is same as left most non-terminal on R.H.S, then such a grammar is called left recursive grammar.

  **Exc**: $A \rightarrow A\alpha / \beta$
- Problem with left recursive grammar is that if such a grammar is used by parser that uses LMD in deriving a string, then there is a danger of going into an infinite loop.
- The eliminate left recursion: For each rule of the form $A \rightarrow Aa / \beta$ rewrite as

$$A \rightarrow \beta A^1,\ A^1 \rightarrow \alpha A^1/\varepsilon$$

- This can be generalized as for each rule

$$A \rightarrow A\alpha_1,\ A\alpha_2\cdots\cdots\cdots / \beta_1/\beta_2\cdots\cdots\cdots$$

  Solution grammar which is free of left recursion is

$$A \rightarrow \beta_1 A^1/\ \beta_2 A^1/\ \cdots\cdots\cdots$$

$$A^1 \rightarrow \alpha_1 A^1 / \alpha_2 A^1/\cdots\cdots\cdots\cdots/\varepsilon$$

- **Left Factoring :** When a rule has common prefix at more than one alternative production, then the parser may not be able to decided which alternative to use.

  **Ex:** $A \rightarrow \alpha\beta_1 /\ \alpha\beta_2$
- Solution to avoid such ambiguity is left factoring. Result of left factoring above grammar is

$$A \rightarrow \alpha A^1 \qquad A^1 \rightarrow \beta_1 /\beta_2$$

- Left factoring avoids backtracking in parser
- Left factoring is not a solution for eliminating ambiguity



For More Study Materials : www.keralanotes.com

**Example 1.1:**

Consider the grammar

S→a S/Sa

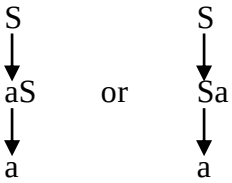
Is this grammar ambiguous ?

How many parse trees exist for an input string w=aaa

**Sol:** Ambiguity

Consider w=aa

i. The derivation trees are




ii. The leftmost derivations are

S ⇒ aS ⇒aa

Or S ⇒ aS ⇒aa

There are two leftmost derivations

iii. The right most derivations are

S ⇒ aS ⇒aa

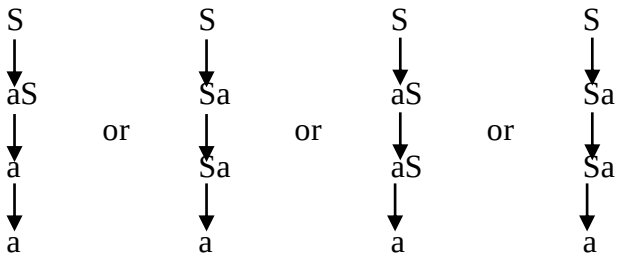
Or S ⇒ aS ⇒aa

There are two rightmost derivations

So the grammar is ambiguous

Consider the sentence w=aaa

The parse trees are




There are 4 derivation trees for w=aaa



For More Study Materials : www.keralanotes.com

**Example 1.2:**

Consider the grammar

S→aSbS/bSaS/ε

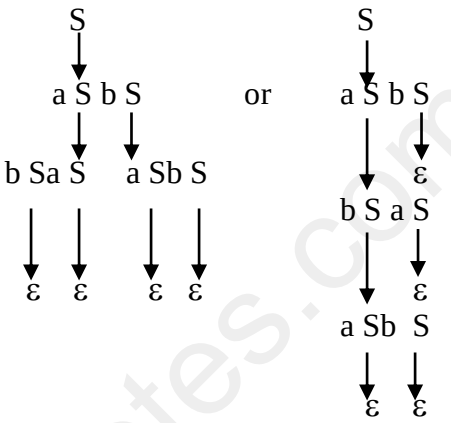
Is this grammar ambiguous ?

How many parse trees exist for w=abab

**Sol:** Ambiguity

Consider the string w=ababab

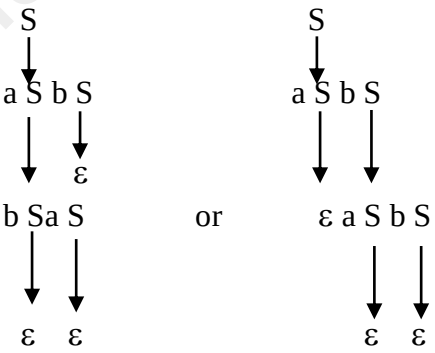
i. The derivation trees are




There are two or more derivation trees for the sentence w=abab,

so the grammar is ambiguous.

Parse trees for w= abab




There are two parse trees for the sentence w=abab

keralanotes.com



For More Study Materials : www.keralanotes.com

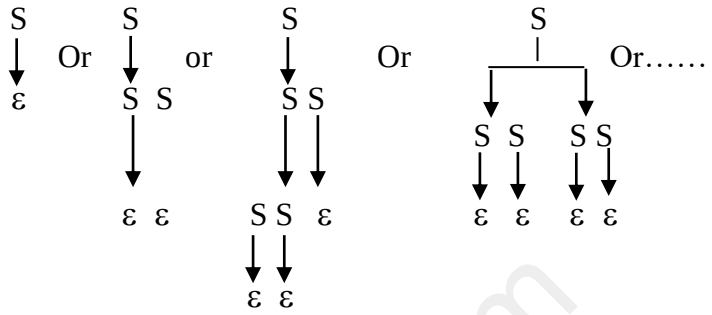
**Example 1.3:**

Check if G is ambiguous

$S \rightarrow aSb/SS/\varepsilon$

**Sol:** Consider the string $w=\varepsilon$

The derivation trees are




There are an infinite number of derivation trees for $w = \varepsilon$.

So the grammar is ambiguous.

**Example 1.4:**

Eliminate left recursion in

$A \rightarrow b/ Bd$

$B \rightarrow Bc/ Ac$

**Sol:** Eliminate the left recursion from the B –rules

$A \rightarrow b/ Bd$

$B \rightarrow AcZ$

$Z \rightarrow \varepsilon /cZ$

Substituting $A \rightarrow b / AcZd$

$B \rightarrow AcZ$

$Z \rightarrow \varepsilon /cZ$

Eliminate the elf recursion from the A-rules

$A \rightarrow bZ_1$

$Z_1 \rightarrow cZ\ d\ Z_1 / \varepsilon$

$B \rightarrow AcZ$

$Z \rightarrow \varepsilon /cZ$

The grammar is not left recursive

keralanotes.com



For More Study Materials : www.keralanotes.com

**Example 1.5:**

Left factor the grammar

$S \rightarrow a\ b\ c\ /\ abd/\ a\ e\ f$

**Sol:** $S \rightarrow a\ [\ b\ c \mid\ b\ d \mid\ \ e\ f\ ]$

**Example 1.6:**

Left factor the grammar

$S \rightarrow a\ S\ S\ b\ S \mid\ \ a\ S\ a\ S\ b \mid a\ b\ b \mid b$

**Sol:** $S \rightarrow a\ [\ S\ S\ b\ S \mid\ Sa\ Sb \mid\ bb\ ] \mid\ b$

Or

$S \rightarrow a\ X \mid b$

$X \rightarrow S\ \ [S\ b\ S \mid a\ S\ b] \mid b\ b$

Or

$S \rightarrow a\ X \mid b$

$X \rightarrow S\ Y\ \ \mid b\ b$

$Y \rightarrow S\ b\ S \mid a\ S\ b$

**Example 1.7:**

Eliminates left recursion

$S \rightarrow SOS\ /\ 1\ 0\ 1$

**Sol:** $S \rightarrow\ 1\ 0\ 1\ Z$

$Z \rightarrow \varepsilon\ /\ OSZ$

## The Role of the Parser

The parser obtains a string of tokens from the lexical analyzer, as shown in fig 1.1, and verifies that the string of token names can be generated by the grammar for the source languages.

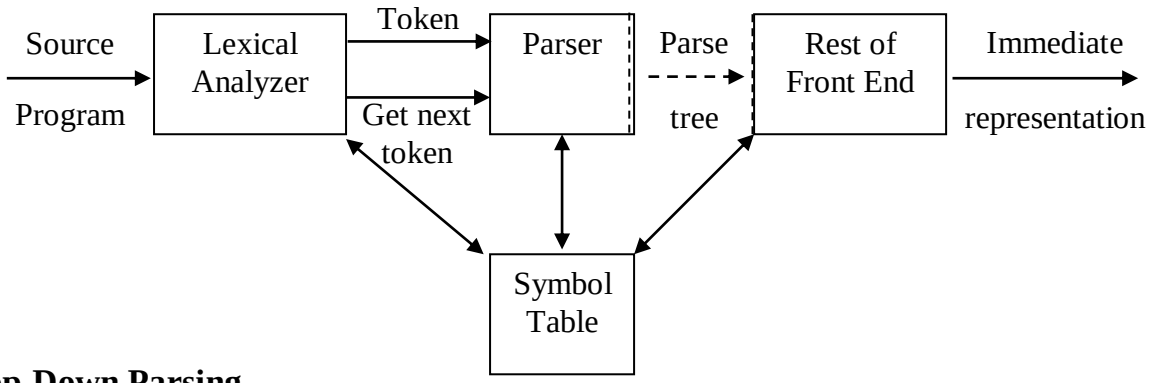
Conceptually, for well –formed programs, the parser constructs a parse tree and passes it to the rest of the compiler for further processing. In fact, the parse tree need not be constructed explicitly, since checking and translation actions can be interspersed with parsing.

keralanotes.com



For More Study Materials : www.keralanotes.com

Thus, the parser and the rest of the front end could well be implemented by a single module.




**Top-Down Parsing**

Top-down parsing can be viewed as the problem of constructing a parse tree for the input string, starting from the root and creating the nodes of the parse tree in preorder, Equivalently, top-down parsing can be viewed as finding a leftmost derivation for an input string.

**Example :** Consider the Grammar

$E \rightarrow T\ E'$

$E' \rightarrow +\ T\ E' \mid \varepsilon$

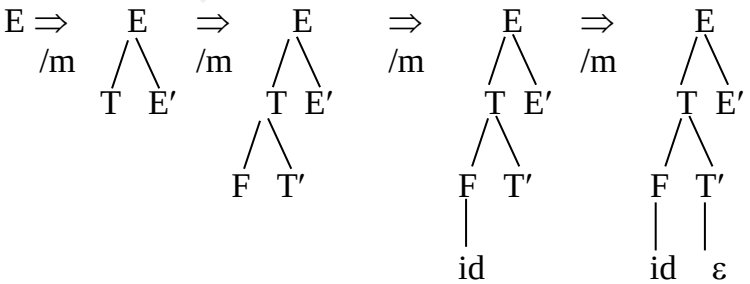
$T \rightarrow +\ F\ T'$

$T' \rightarrow *\ F\ T' \mid \varepsilon$

$F \rightarrow (E) \mid id$

Generating the top-down parse for the input string id+id *id.

The following sequence of trees corresponds to a leftmosh(/m) derivation of the input






For More Study Materials : www.keralanotes.com

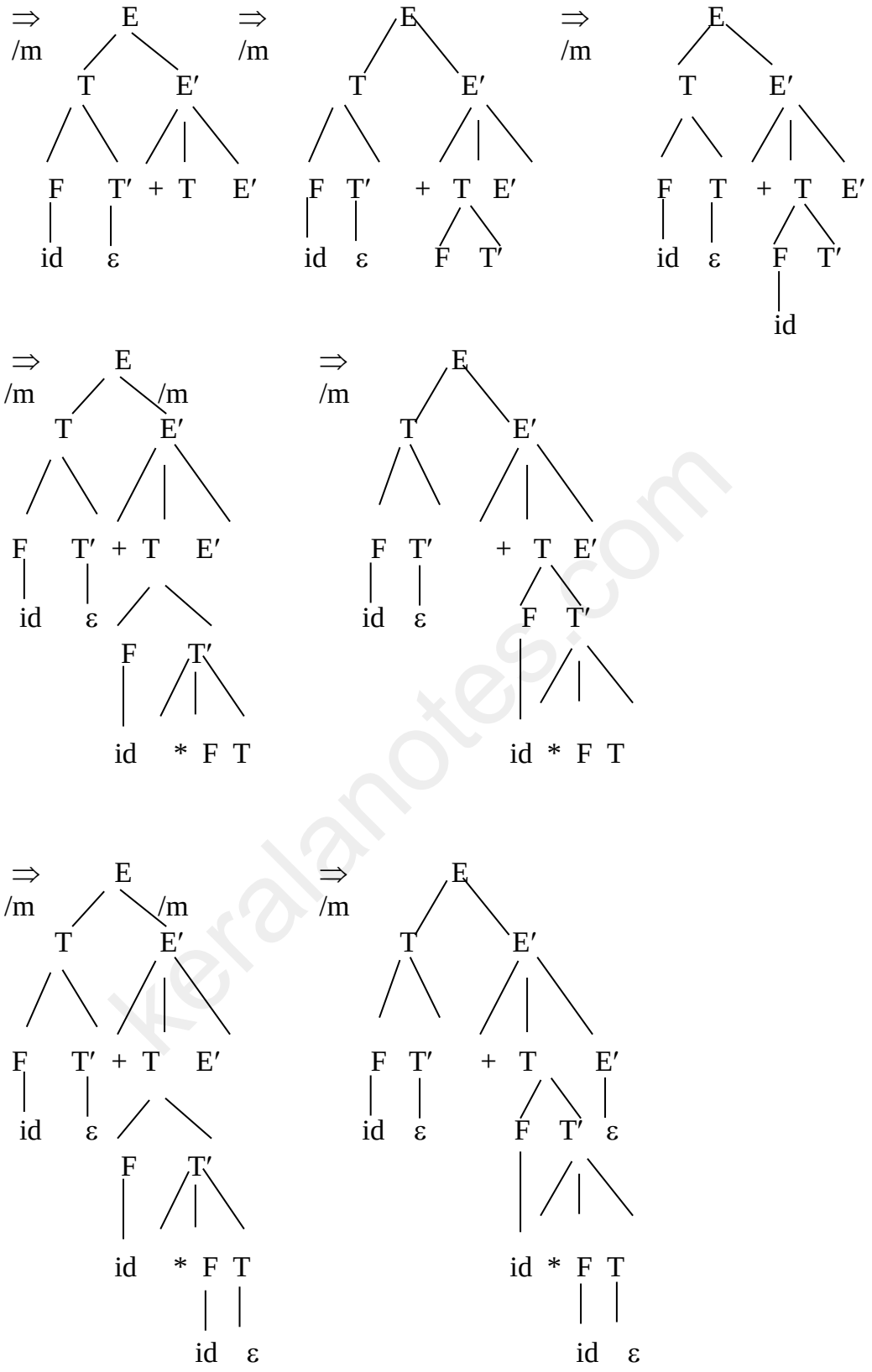



**Fig 1.2: Top-down parse for id + id * id**

keralanotes.com



For More Study Materials : www.keralanotes.com

**Classification of Top-Down parsers:**

- Recursive Descent Parsing
- LL(I) Grammars
- Non-recursive predictive parsing

**Bottom Up Parsing**

A bottom up parse corresponds to the construction of a parse tree for an input string beginning at the leaves (the bottom) and working up towards the root (the top). It is convenient to describe parsing as the process of building parse trees, although a front end may in fact carry out a translation directly without building an explicit tree.

**Example:** For above mentioned grammar generate the bottom up parse for input string id*id.

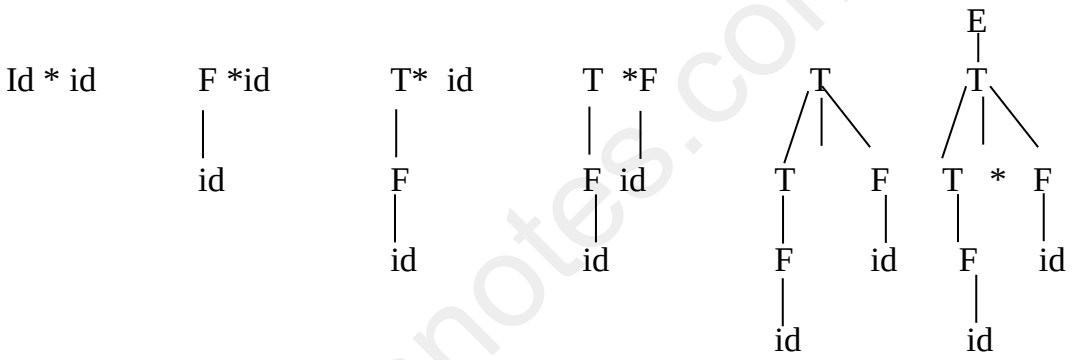



**Fig 1.3: A bottom –up parse for id *id**

**Classification of Bottom up parsers:**

- Shift Reduce parsing
- Operator precedence parser
- LR(k) Parsing
  - LR(O)
  - SLR(I)
  - LALR(I)
  - CLR(I)

keralanotes.com



For More Study Materials : www.keralanotes.com

## SYNTAX DIRECTED TRANSLATION

- SDT – Generates code while parsing by following rules attached with each production.

  **Ex:** A →α Printf ("Hai");

  Whenever parser reduces by A →α gives message"Hai"

- **Synthesized attribute:** – If attribute value at a node is defined in terms of attribute at its children, such an attribute is called synthesized attribute

  **Ex: m** A → BC, if A. a = f (B.a. or C.a)

  Then attribute 'a' is synthesized

- **Inherited attribute:** – If attribute value at a node is defined in terms of attributes at its parent or sibling, such an attribute is called inherited attribute

  **Ex:** In rule A → BC, if

  B. I = f(A. i or C. i); then 'i' is called inherited attribute

- **Annotated /Decorated parse tree:** – A parse tree which shows attribute values at each node is called annotated or Decorated parse tree
- **S-attribute definition:** An SDT which uses only synthesized attribute is called S-attributed definition.
- **L-attributed definition:** An SDT on which attribute are restricted to inherit either from parent or left siblings is called L-attributed definition.
- Every S-attributed definition is L-attributed

**Example 1.8:**

Write a SDT for evaluating an infix expression.

**Sole:** $E \rightarrow E_1 + E_2$ { E.val = $E_1$.val+$E_2$.val}

Out (E.val)

/$E_1$*$E_2$ {E.val = $E_1$. val + $E_2$.val}

/ ($E_1$) {E.val= $E_1$.val}

/num {E.val= num.val}



For More Study Materials : www.keralanotes.com

**Example 1.9:**

Write a SDT to convert infix to postfix notation

**Sole:** $E \rightarrow E_1 + E_2$ { E.val = $E_1$.val+$E_2$.val+}

/$E_1$*$E_2$ {E.val = $E_1$. val $E_2$.val+}

/ $E_1$* $E_2$ {E.val= $E_1$.val $E_2$. val*}

/($E_1$) {E.val = $E_1$.val}

/id {E.val= id.name}

## Intermediate Code Generation

- Advantage of intermediate code generation is
  - Retargeting is supported
  - Machine independent code optimization can be applied
- I.C.G. can be represented as
  - Syntax tree
  - DAG
  - Post-fix notation
  - Three address code
- Three address code can be implemented as
- **Quadruple** – a record with 4 fields ( oprd, oprd1, oprd2, result)

  **Advantage -** Statements can be moved around.

  **Disadvantage-** wastage of space for temporaries
- **Triples** – a record with 3 fields ( oprd, oprd1, oprd2 )

  **Advantage -** no wastage of space

  **Disadvantage-** but statements cannot be moved around.
- **In-direct Triples** – maintains list of pointers to triples

  **Advantage -**

  1. Statements can be moved around
  2. No wastage of space for temporaries
- **Types of 3 address statements**

  1. x = y op x
  2. x = op y
  3. x= y [i]

keralanotes.com



For More Study Materials : www.keralanotes.com

4. x[i] = y
5. x = *y
6. x= &y
7. if x reloop y goto L
8. goto L

**Example 1.10:**

```
While a < b do
If      c < d then
        x : = y +z
else
        x : = y  - z
Lbegin: if a < b goto L1
        goto Lenext
L1:     if c < d goto L2
        goto L3
L2:     tl: = y +z
        x : = t1
        goto Lbegin
L3:     t2:= y – z
        x: = t2
        goto Lbegin
Lnext:
```

## Runtime Environments

**Static and Dynamic Notions**

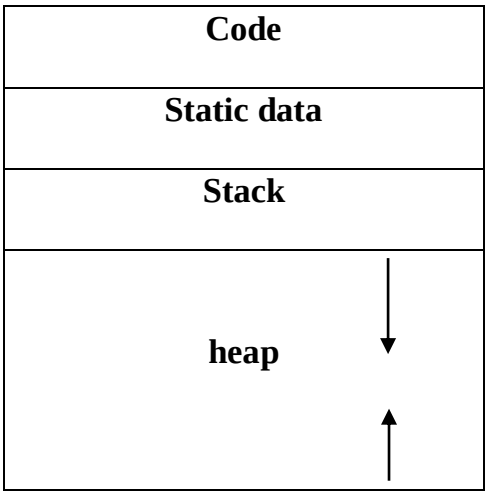
| | |
|---|---|
| Static | Definition of a procedure<br>Declaration of a name<br>Scope of a declaration |
| Dynamic | Activation of a procedure<br>Binding of a name<br>Lifetime of a binding |

keralanotes.com



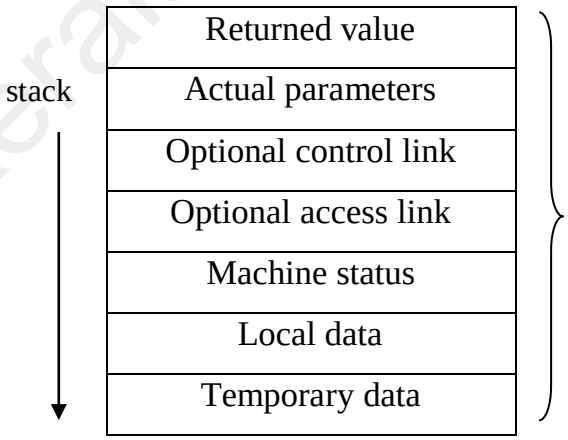
For More Study Materials : www.keralanotes.com

**Storage Organization**




- Target code: static
- Static data objects: static
- Dynamic data objects: heap
- Automatic data objects: stack
- Information needed for each execution of a procedure is stored in a record called activation record.
- Activation record contains 7 fields

**Activation Records**




Size of each of this can be estimated at compile time

- Before Target code generation, we need to consider how programs can be implemented on target machines in particular
- How are data objects represented memory

keralanotes.com



For More Study Materials : www.keralanotes.com

- How is storage of data objects be allocated
- How are parameters passed to function
- Answers to this are very much depended on storage allocation strategies
- **Static Allocation:** Storage is allocated at compile time only
- Static storage has fixed allocation that does not change during program execution.
- As bindings do not change at runtime, no runtime support is required.

**Disadvantages:**

1. Size of data objects should be known at compile time
2. Recursion is not supported
3. Data structures cannot be created at runtime

**Stock Allocation:**

- is based on idea of control stack
- activation record are pushed and popped as activations begin and end respectively.
- Locals are always bound to fresh storage in each activation because a new activation is pushed onto stack when a call is made.
- Values of locals deleted as activation end.

**Disadvantages**

- Values of locals cannot be retained once activation ends.

**Heap Allocation**

- Storage can be allocated and de-allocated in any order
- So ever a time, it may contain pieces of alternate areas that are free and in use

**Disadvantage**: Heap manager overhead.

## BASIC OF OPTIMIZATION

**Optimization refers to technique used by compiler to improve the execution efficiency of the generated code.**

- Every optimization should preserve semantics
- Actually after applying optimization there is no guarantee that generated code is optimal



For More Study Materials : www.keralanotes.com

- Optimizations that are restricted to one basic block are called **local optimizations;** Otherwise, they are called **global optimizations.** Compiler optimizations can be broadly divided into two categories.
- Machine dependent Optimizations- optimizations that can be applied on target code.
- Machine independent optimizations – optimizations that can be applied on three address code.

**Examples of Machine independent optimizations are:**

1. Loop optimization
2. Folding
3. Constant Propagation
4. Copy Propagation
5. Redundancy elimination
6. Strength reduction
7. Dead-Code eliminations
8. Algebraic simplifications

**Examples of Machine dependent optimizations are:**

- Peephole optimization
- Register allocation
- Effective usage of addressing modes.

**Machine independent optimization**

1. **Loop optimization:** Loops are principle sources of optimization as computer spends much time. These act on the statements which make up a loop, such as a for loop. Loop optimizations can have a significant impact because many programs spend a large percentage of their time inside loops.

   For detecting loops control flow analysis ( CFA) is used. In CFA, a program is represented by program flow graph ( PFG) which shows how control is flowing in a program.

   To obtain such graph, intermediate code is divided into basic blocks.

keralanotes.com



For More Study Materials : www.keralanotes.com

**Control Flow Graphs**

- A (control) flow graph is a directed graph
- The modes in the graph are basic blocks
- There is an edge from B1 to B2 iff B2 immediately follows B1 in some execution sequence.
  - There is a jump from B1 to B2
  - B2 immediately follows B1 in program text
- B1 is a predecessor of B2, B2 is a successor of B1

**Basic Blocks**

- A basic block is a sequence of consecutive statements in which control enters at the beginning and leaves at the end without halt or possibility of branching except at the end.

**Construction of Basic Blocks**

- Determine the set of leaders
  - The first statement is a leader
  - The target of a jump is a leader
  - Any statement immediately following a jump is a leader
- For each leader, its basic block consists of the leader and all statements up to but not including the next leader or the end of the program.

**Representation of Basic Blocks**

- Each basic block is represented by a record consisting of
  - a count of the number of statements
  - a pointer to the leader
  - a list of predecessors
  - a list of successors

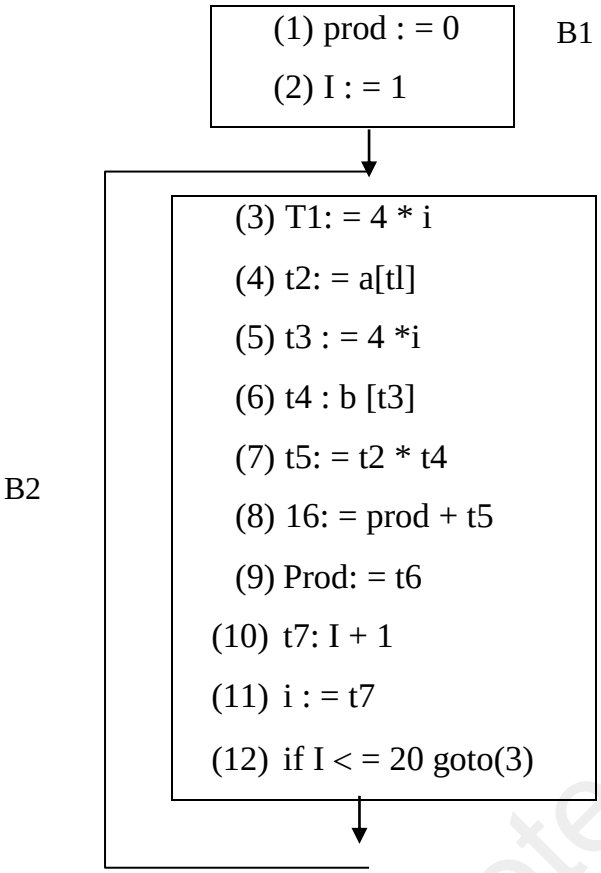
**An example**

1) Proud: = 0
2) i : = 1
3) t1: = 4 * i
4) t2: = a[t1]
5) t3: = 4 * i
6) t4: = b[t3]
7) t5: = t2*t4
8) t6:=prod +t5
9) prod:=t6
10) t7:=i+1
11) I : = t7
12) If i < = 20 goto (3)



For More Study Materials : www.keralanotes.com

PFG is




**Loop detection:**

A loop is a cycle in flow graph

**Some of Loop optimizations are :**

(a) Code motion / loop invariant computation elimination / frequency reduction:

This involves moving loop invariant computation outside the loop as follows.

```
While ( i < = 1000)
        x = SIN(A)*COS(A)*i
```

as

```
        t = SIN(A)*COS(A);
        while (i < 1000)
                x=t*i ;
```

(b) **Loop unrolling:** Unrolling duplicates the body of the loop multiple times, in order to decrease the number of times the loop condition is tested and the number of jumps, which hurt performance by impairing the instruction pipeline. Completely unrolling a loop eliminates all overhead, but requires that eh number of iterations be known at compile time.



For More Study Materials : www.keralanotes.com

Ex: while ( i <100)

{

x[i]=0;

i++;

}

as

while ( i <100)

{

x[i]=0;

i++;

x[i]=0;

i++;

}

There is clearly a trade off space Vs Time.

(c) **Loop jamming** / Loop fusion or loop combining another technique which attempts to reduce loop overhead. When two adjacent loops would iterate the same number of times ( whether or not that number is known at compile time), their bodies can be combined as long as they make no reference to each other's data.

2. **Folding / constant folding :** Replacing expressions consisting of constants (e.g. "3+5") with their final value ("8") at compile time, rather than doing the calculation in run-time. Used in most modern languages.

3. **Constant Propagation :** Replacing variables by their associated values known at compile time.

**Ex:** pi =3.1415

D_to_R=pi/180=3.1415/180

4. **Copy propagation :** Propagate copy may be useful in finding dead code.

**Ex:**

a. x=y

b. z=1.0+x! dependent on previous "x"

- Problem because no parallelism! no pipelining!
- Second statement depends on last



For More Study Materials : www.keralanotes.com

- Would be better to write
- x =y
- z= 1.0 +y

**5 Dead Code Removal :**

- Blocks that are never entered can be removed
- Not much use to programmer
- But during multiple passes compiler may generate dead code.

**Dead Code elimination:** Removes instructions that will not affect the behaviour of the program, for example definitions which have no uses, called dead code. This reduces code size and eliminates unnecessary computation.

**6. Common sub expression elimination / Redundancy elimination :** In the expression "(a+b)- (a+b)/4", "Common sub expression" refers to the duplicated "(a+b)". Compilers implementing this technique realize that "(a+b)" won't change, and as such, only calculate its value once.

Dags are also useful in redundancy elimination.

**7 Strength Reduction :** Replace a costlier operation by cheaper one.

- Classic one is exponentiation:
- x**2= x*x
- Exponentiation is expensive (software)
- Help the compiler here:

**8. Algebraic simplification :** eliminate statements like x=x*1 or y = y +0;

**DAG Representation of Blocks**

- Easy to determine:
- Common sub expressions
- Names used in the block but evaluated outside the block
- Names whose values could be used outside the block
- Leaves labeled by unique identifiers
- Interior nodes labeled by operator symbols
- Interior nodes optionally given a sequence of identifiers, having the value represented by the nodes.



For More Study Materials : www.keralanotes.com

**An Example**

(1) t1 := 4*i

(2) t2: =a[t1]

(3) t3:= 4*i

(4) t4: = b[t3]

(5)t5:t2*t4

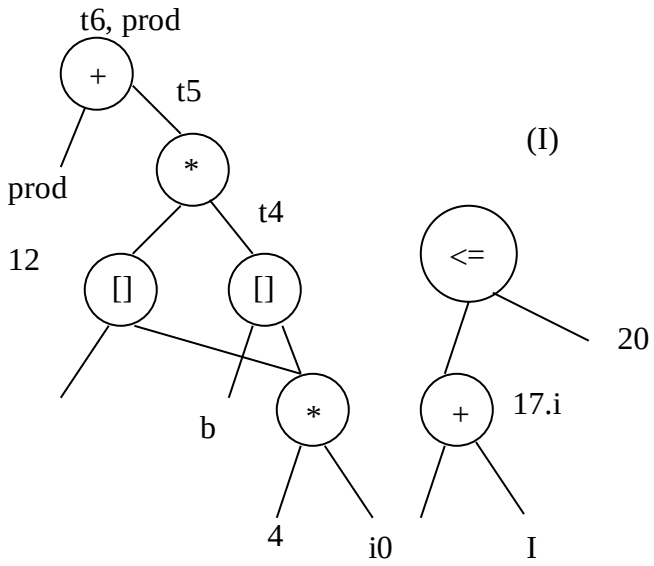
(6) t6:prod+t5

(7)prod: t6

(8) t7: = i+1

(9) i:= t7

(10) if i<= 20 goto(1)




**Generating Code from DAGs**

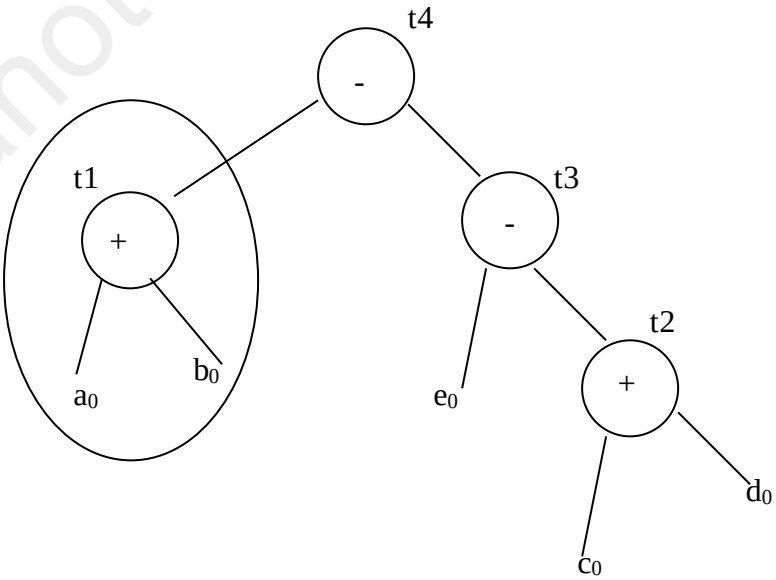
t1: = a+b

t2: + c+d

t3: = e – t2

t4: = t1 –t3

1) MOV a, R0
2) ADD b, R0
3) MOV c, R1
4) ADD d, R1
5) MOV R0, t1
6) MOV e, R0
7) SUB R1, R0
8) MOV t1, R1
9) SUB R0, R1
10) MOV R1,t4

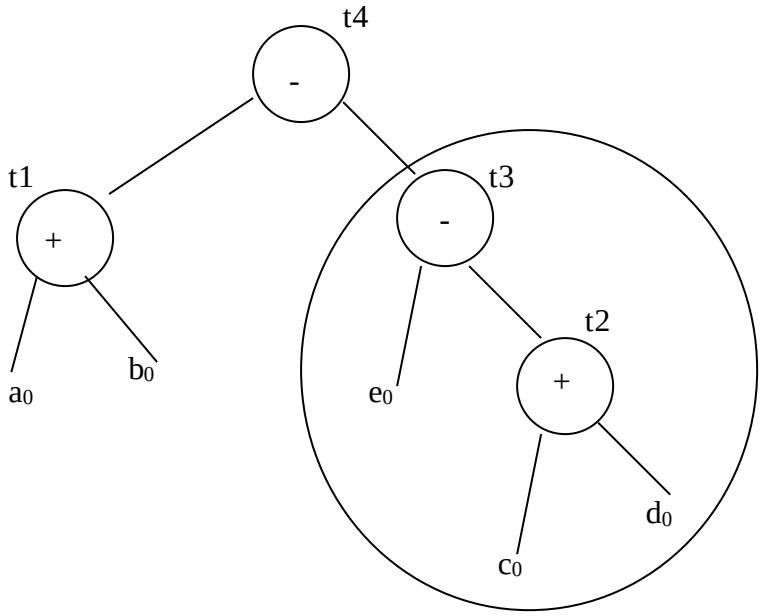



Only R0 and R1 available

keralanotes.com



For More Study Materials : www.keralanotes.com




**Rearranging the order**

t2: + c+d

t3: = e – t2

t1: = a +b

t4: = t1 –t3

1) MOV c, R0
2) ADD d, R0
3) MOV e, R1
4) SUB R0, R1
5) MOV a, R0
6) ADD b, R0
7) SUB R1, R0
8) MOV R0,t4

**Machine Dependent Optimization:**

**Peephole Optimization**

- Improve the performance of the target program by examining and transforming a short sequence of target instructions.
- May need repeated passes over the code
- Can also be applied directly after intermediate code generation

**Examples**

(a) **Redundant loads and stores**

MOV R0, a

MOV a, R0

(b) **Algebraic Simplification**

x : = x +0

x : = x * 1

(c) **Unreachable code**

#define debug 0

If (debug) ( print debugging information)

If 0<>1 goto L1

Print debugging information

keralanotes.com



For More Study Materials : www.keralanotes.com

L1:

If 1 goto L1

Print debugging information

L1:

(d) **Flow-of-control** **optimization**

| | |
|---|---|
| goto L1 | goto L2 |
| … | …. |
| L1: goto L2 | L2.goto L2 |
| goto L1 | if a < b goto :2 |
| … | goto L3 |
| L1: if a < b | goto L2 |
| … | |
| L3: | L3: |

(e) **Reduction in strength:** replace expensive operations by cheaper onces

- $x^2 \Rightarrow x^* x$
- fixed point multiplication and division by a power of 2 $\Rightarrow$ shift
- floating-point division by a constant $\Rightarrow$ floating –point multiplication by a constant.

(f) **Use of machine idioms:** hardware instructions for certain specific operations auto-increment and auto-decrement addressing mode (push or pop stack in parameter passing).

keralanotes.com



For More Study Materials : www.keralanotes.com

# CHAPTER – 2

# LEXICAL ANALYSIS

## Concepts & Definitions

1. Tokens – The lexical analyzer (sometimes called scanner) is the front end of the compiler. It breaks up the source program into lexemes & outputs them as a stream of tokens. Wherever the syntax analyzer calls the scanner it gets a token in return.
2. Lexemes – Every token has an associated lexeme the sequence of input character that the token represents.
3. Buffers – It is sometimes necessary for the scanner to scan past the lexeme into the next lexeme. To facilitate this buffering is used.
4. Pattern- Each token has an associated pattern normally regular expressions are model patterns.
5. Regular expression-An algebra which has union, concatenation & kleene closure as operators used to model the structure of lexemes & tokens.
6. Regular Definition – Regular grammar giving the set of rules of the formation of tokens.
7. Extended Regular Expressions – includes + for one or more & ? for zero or one of operation.
8. Transition diagrams – formed from regular expression also called finite automata with $\in$ moves.
9. Finite automata – Model the scanner
10. Deterministic Finite automata – No choice in any situation. Accept the regular sets described by regular expression.
11. Non Deterministic Finite automata – have choice & move naturally model regular definitions.
12. Lex-A'C' tool for automatic generation of lexical analyzer.

keralanotes.com



For More Study Materials : www.keralanotes.com

**2.1 Definition**

The scanner or lexical analyzer is the front end of the compiler. It automatically generates a listing by printing out whatever is read in. Preprocessors by macros are handled by lexical analyzer.

**2.2 Concept**

The scanner breaks up the input into lexemes. These are mapped into tokens.

**2.3 Strategy**

The structure of identifiers, reserved words, operators, number & comments can be described by a regular expression / regular grammar.

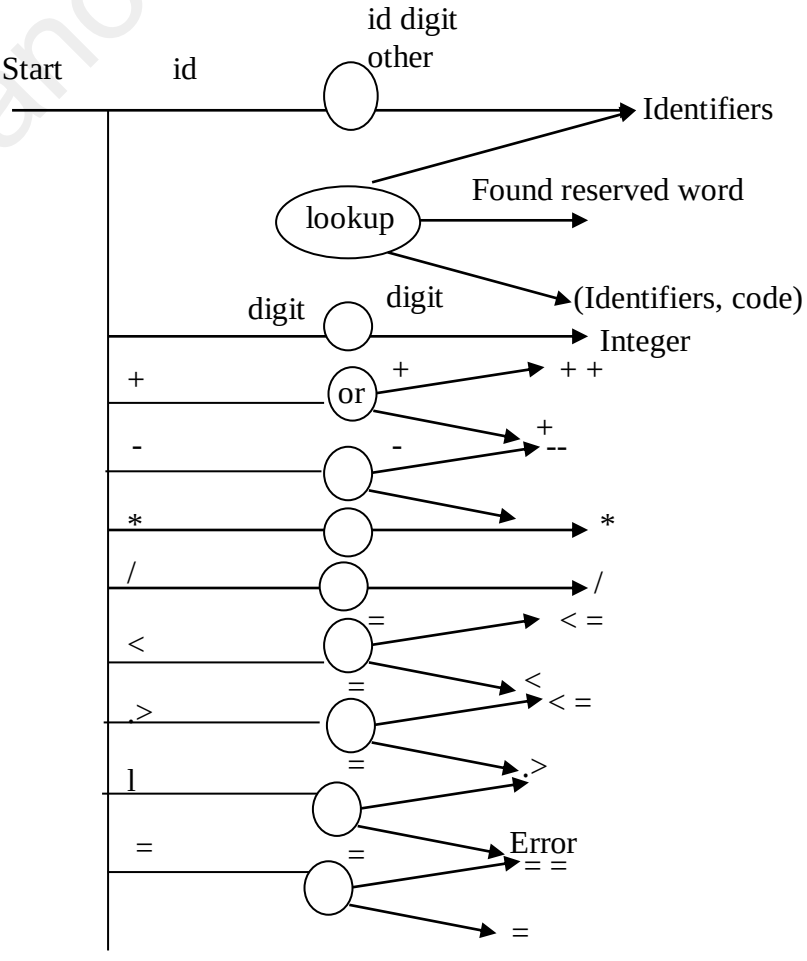
**2.4 Concept**

A lexical analyzer is modeled by a transition diagram of a finite automata .

**2.5 Concept**

One can specify the structure of lexemes by a regular expression & then from the regular expressions build a deterministic finite automata which is the lexical analyzer.

**2.6 Structure of the lexical analysis**




keralanotes.com



For More Study Materials : www.keralanotes.com

**2.7** We will use a running a example throughout the various phases of the compiler as follows:

1. Description of the running example – quick sort
2. Breakup of the running example into lexemes
3. Breakup of the running example into tokens
4. A cfg for the running example
5. A parse tree for the running example

6 The SDTS for the running example

7 Translation of the running example to intermediate 3-address code

8 Translation of the running example to optimized 3 – address code

9 The code for the running example in a hypothetical machine.

**2.8** The running example chosen.

```
Void quicksort (int m, int n)
/*recursively sort the elements a[m] through a[n]*/
{
        int I, j;
        int v, x;
        if(n<=m)return;
        I =m-1; j=n; v= a[n];
        While (I) {
                do i = I +1; while (a[i]<v);
                do j = j –I; while (a[j]>v);
                if (i>= j) break;
                x=a [i], a[i]= a[j];a[j] = x;
                /*swap a[i], a[j]*/
        }
        x= a[i]; a[i]=a[n]; a[n]=x;
                /*position partition element */
        Quicksort(m.j); quicksort(i+1,n);
```

keralanotes.com



For More Study Materials : www.keralanotes.com

**2.9** Classification of a running example into lexemes & tokens

| | | | | | |
|---|---|---|---|---|---|
| void | quicksort | ( | int | m | , |

| | | |
|---|---|---|
| int | n | ) |

Comment is removed

| | | | |
|---|---|---|---|
| j | = | n | ; |

| | | | | | | |
|---|---|---|---|---|---|---|
| v | = | a | [ | n | ] | ; |

| | | | | |
|---|---|---|---|---|
| while | ( | I | } | { |

| | | | | | | |
|---|---|---|---|---|---|---|
| do | i | = | i | + | I | ; |

| | | | | | |
|---|---|---|---|---|---|
| while | ( | a | [ | i | j |

| | | | |
|---|---|---|---|
| < | v | ) | ; |

| | | | | | | |
|---|---|---|---|---|---|---|
| do | j | = | j | - | I | ; |

| | | | | | |
|---|---|---|---|---|---|
| while | ( | a | [ | j | ] |

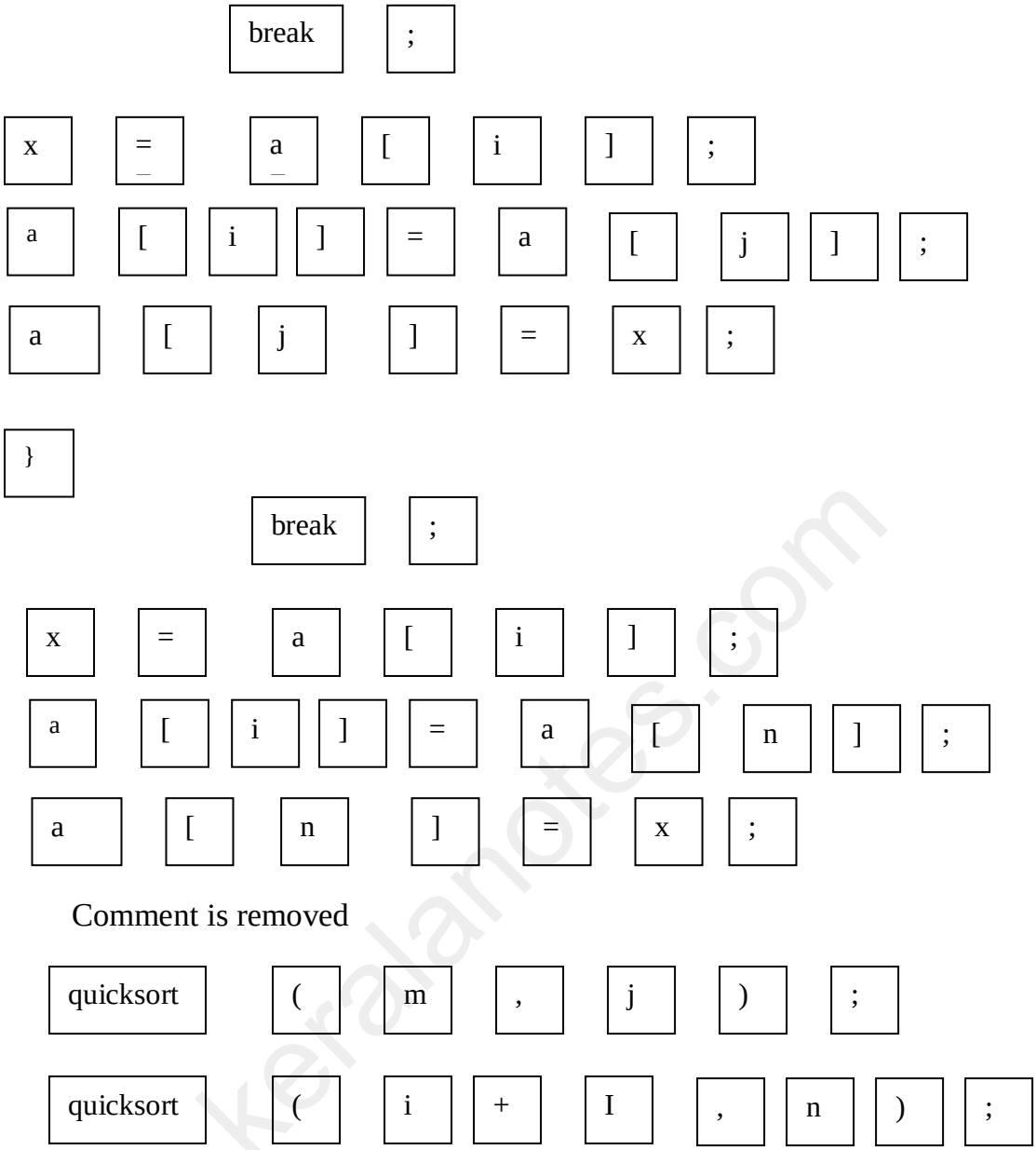
| | | | |
|---|---|---|---|
| > | V | ) | ; |

| | | | | | |
|---|---|---|---|---|---|
| if | ( | i | >= | j | ) |

keralanotes.com



For More Study Materials : www.keralanotes.com




## 2.10 Symbol table

| | Name | Type | Width |
|---|---|---|---|
| 1 | i | int | 4 |
| 2 | j | int | 4 |
| 3 | v | int | 4 |
| 4 | x | int | 4 |
| 5 | n | int | 4 |
| 6 | a | Array | |
| 7 | Quicksort | Function | Parameters |
| 8 | m | int | 4 |
| 9 | n | Int | 4 |



For More Study Materials : www.keralanotes.com

**2.11** Constant table

| | **Name** | **Value** |
|---|---|---|
| 1 | "1" | I |

**2.12** Reserved words:

| | | |
|---|---|---|
| 10 | **Void** | |
| 11 | If | I |
| 12 | do | |
| 13 | while | |
| 14 | Return | |
| 15 | Break | |
| 16 | Int | |

**2.13** token string of the running example

10 7 (16 var I, 17 var 5)

{

16 var I, var 2;

16 var 3, var 4;

11 (var n < = var 8) 14;

Var 1 = var 8 – cons 1; var 2 = var 5;

Var3 = var 6 [ var 5] ;

13 (var 1) { 12 var 1 = var 1 + const 1;

13 (var 6 [var 1] < var3);

12 var = var 2 + const1;

13 (var 6 [ var 2 ≥ var 3);

11 (var 1> = var 2) 15;

var 4 = var 6 [ var 6 [var 2];

var 6 [ var 2] = var 4;

}

keralanotes.com



For More Study Materials : www.keralanotes.com

var 4 = var 6 [ var 1];

var 6 [var 1] = var [var 5];

var 6 [ var 5] =v ar 4;

var 7 ( var 8, var 2) ;

var 7 (var 1+const 1, var 5);

**Example 2.1:**

How many lexemes are there in the following statement ?

Print ("total = %d/n ", score);

(a) 4 (b) 5 (c)6 (d) 7

Ans: (d)

Sol :

| printf | ( | "Total = % d/n" | | | |
|---|---|---|---|---|---|
| , | Score | ) | ; | | |

**Example 2.2:**

How many different tokes are there in the following statement ?

Print ("total = %d/n ", score);

(a) 4 (b) 5 (c)6 (d) 7

Ans: (c)

Sol :

| id | ( | "Total = % d/n" | | | |
|---|---|---|---|---|---|
| , | id | ) | ; | | |

**Example 2.3:**

How many lexemes are there in the following FORTRAN statement ?

DO 51 = 1.25

(a) 8 (b) 5 (c)3 (d) 4

Ans: (c)

Sol :

| DO 51 | = | 1.25 |
|---|---|---|

keralanotes.com



For More Study Materials : www.keralanotes.com

**Example 2.4:**

How many lexemes are there in the following FORTRAN statement ?

DO 51 = 1.25

(a) 5 (b) 3 (c)8 (d) 7

Ans: (d)

Sol :

| DO | 5 | 1 | = | 1 | , | 25 |
|---|---|---|---|---|---|---|

**Example 2.5:**

How many different kinds of tokens are there in the following FORTRAN statement ?

DO 5J = 1,250

(a) 5 (b) 6 (c)4 (d) 8

Ans: (c)

Sol :

| int | num | int | = | num | , | num |
|---|---|---|---|---|---|---|

Token types

(1) int (2) num (3) = (4) ,

**Class Practice Questions**

01. Comments are normally detected from the source program during

a) Lexical analysis b) Syntax analysis

c) Semantic analysis d) Code generation

02 The macro –expansion or expansion of preprocessor statements take place during

a) Lexical analysis b) Syntax analysis

c) Semantic analysis d) Code optimization

03 Keywords are recognized in a compiler during

a) Lexical analysis b) Syntax analysis

c) Semantic analysis d) Code generation

04 Type checking is done during

a) Lexical analysis b) Syntax analysis

c) Semantic analysis d) Code generation

keralanotes.com



For More Study Materials : www.keralanotes.com

05 Compiler can diagnose the following errors ?

a) Grammatical errors b) Logical errors

c) Runtime errors d) All of the above

06 A compiler that may run on one machine and produces target code for another machine is called _____

a) Bootstrapping b) Direct Compiler

c) Indirect compiler d) Cross compiler

07 The object code which is obtained from Assembler is

a) Assembly language code b) Relocatable object code

c) Executable object code d) None

08 Choose the correct statement ?

a) A compiler must compulsorily keep lexical analysis separate from the syntax analysis.

b) The syntax analysis phase can be expanded to do lexical analysis also

c) Macro expansion is done during syntax analysis

d) Macro expansion is done during semantic analysis.

09 Choose the correct statement ?

a) The symbol table is constructed in the lexical analysis phase

b) The symbol table is constructed during the syntax phase.

c) The symbol table is not needed for semantic analysis

d) The symbol table is not needed for code generation.

10 Choose the correct statement ?

a) Most of the time for compilation is spent in lexical analysis

b) Most of the time for compilation is spent in semantic analysis

c) Most of the time for compilation spent during code optimization phase

d) Most of the time for compilation is spent during the code generation phase.

11 Choose the correct statement ?

a) The lex utility generates a lexical analyzer

b) The lex utility cannot handle all regular expressions

c) The lex utility cannot handle all regular definitions

d) None of the above.



For More Study Materials : www.keralanotes.com

12 The output of the Lex compiler is

a) Lex, yy b) Lex, yy.c

c) a,out d) Tokens

13 Consider the following error recovery operations in lexical analysis

1) Delete one character from the remaining input

2) Insert a missing character into the remaining input

3) Replace a character by another character

4) Transpose two adjacent character

5) If parenthesis mismatch delete the offending parenthesis

a) 1,2,3,4 are valid at the lexical analysis phase

b) 4,5 cannot be done at the lexical analysis phase

c) 1,2 are not valid at the lexical analysis phase

d) Only 5 is not valid at the lexical analysis phase.

14 for (I = 0; <10; i++)

printf("I = %d", i);

The number of tokens in the above 'C' statement is _____

15 printf ("Total= %d", sum);

The number of tokens in the above 'C' statement is _____

16 Find number of tokens in the following program ?

If ( x>=y ) ( x=x+y;} else {x=x-y;};

a) 23 (b) 24 c) 25 d) None

17 Which of the following is not a token of 'C' program ?

a) .02e+2 b) MAX

c) 123.33 d) None

18 Match the following according to input ( from the left column) to the compiler phase ( in the right column) that processes it:

| | |
|---|---|
| (P) Syntax tree | (i) Code generator |
| (Q) Character stream | (ii)Syntax analyzer |
| (R) Intermediate representation | (iii) Semantic analyzer |
| (S) Token stream | (iv) Lexical analyzer |

keralanotes.com



For More Study Materials : www.keralanotes.com

a) P →(ii), Q→ (iii), R → (iv), S → (i)

b) P → (ii), Q → (i), R → (iii) S → (iv)

c) P → ( iii), Q → (iv), R → (i), S→ (ii)

d) P → (i), Q → (iv), R → (ii), S → (iii)

**KEY**

01.(a) 02.(a) 03.(a) 04.(c) 05.(a) 06.(d) 07.(b)
08.(d) 09.(d) 10.(b) 11.(a) 12.(a) 13.(b) 16.(b)
17.(d) 18.(c)

keralanotes.com



For More Study Materials : www.keralanotes.com

# CHAPTER – 3

# PARSING TECHNIQUES

## Concepts & Definitions

- Parser – A parser takes the stream of tokens given by the lexical analyzer and builds the derivation trees for the source program in accordance with the rule of a context free grammar.
- Context Free Grammar – Also called BNF are a formation to model the syntax or form of most HLLs.
- Terminals – Symbols – in a context free grammar actual occurrence in the language being described.
- Non-Terminals – Meta symbols of a context free grammar used to describe the HLL
- Parse Trees - also called derivation trees and are in accordance with the rules of a context free grammar
- Ambiguity – If two distinct parse trees exist for a string in the language then the grammar is said to be ambiguous.
- Top – down parsing – simulates the left most derivation of a sentence in a context – free grammar.
- Bottom – up parsing – simulates the reverse of a right most derivation in a context – free grammar.
- Design of grammars – Top down parsing grammar must eliminate left recursion and are harder to design.
- Recursive Decent parser – Top down with recursive decent.
- LL(I) Parser – Top down parser with one symbol lookahead
- Shift –reduce parsers –bottom up parser with shift meaning a push operation on the parse start & reduce being a pop.

### 3.1 Concept

In syntax analysis we are interested in determining if the source language is in accordance with the rules of the programming language.



For More Study Materials : www.keralanotes.com

**3.2 Concept :**

We model ( a superset of) the correct programs by a context –free grammar. The problem of syntax analysis reduces to the membership problem of CFLs.

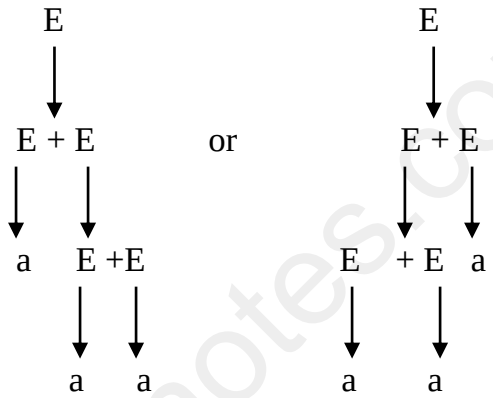
**3.3 Theorem:**

An arbitrary CFG can be put into the Chomsky Normal Form & the membership problem, i.e. syntax analysis can be done by the CYK algorithm. The CYK algorithm is a dynamic programming algorithm of time complexity O ($n^3$).

**3.4 Definition**

An arbitrary CFG may be ambiguous
For example the grammar E $\rightarrow$ E + E/ a is ambiguous

a) The sentence a +a + a has two derivation




If any sentence in the language generated by the grammar has more than one derivation tree then the grammar is ambiguous

b) The sentence a+a+ has two leftmost derivations in the grammar

$E \Rightarrow E + E \Rightarrow E+ E+ E$

$\Rightarrow a + E + E \Rightarrow a+ a +E$

$\Rightarrow a+ a+ a$

$E \xrightarrow[leftmost]{} a+ a+ a$

Or

$E \Rightarrow E + E \Rightarrow a+ E$

$\Rightarrow a + E + E \Rightarrow a+ a +E$

$\Rightarrow a+ a+ a$

$E \xrightarrow[leftmost]{} a+ a+ a$

If a sentence in the language generated by the grammar has more than one right most derivation tree then the grammar is ambiguous



For More Study Materials : www.keralanotes.com

**3.5 Definition**

A grammar is said to be ambiguous if there is more than one derivation tree for a sentence or there is more than one left most derivation for some sentence or there is more than one rightmost derivation for some sentence.

**3.6 Definition**

The grammar $S \rightarrow SS/a/\varepsilon$ has

a) an infinite number of derivation trees for the sentence ε

b) an infinite number of leftmost derivations for the sentence ε

c) an infinite number of rightmost derivations for the sentence ε

**3.7 Definition**

A context –free language is said to be inherently ambiguous if every grammar for it is ambiguous

Example: The language $L = \{a^n b^n c^n d^m / n,m \geq 1\}$ is inherently ambiguous.

**3.8 Definition**

Passing is the systematic construction of a derivation tree to determine if a given sentence is I the language.

**3.9 Definition**

The grammar has a rule $A \rightarrow A$……..

Where A is the non-terminal on the left hand side & the rule is also the first symbol o the right hand side of the rule then the grammar is said to be left recursive.

**3.10 Definition**

If a grammar has a rule $A \rightarrow A$ where A is the non-terminal on the left hand side of the rule is also the last symbol on the right hand side of the rule then the grammar is said to be right recursive.

**3.11 Definition**

If a grammar has no $\in$ - rules i.e. rules of the form $A \rightarrow \in$ and no rule of the form $A \rightarrow$ ….BC… where B, C & A are non-terminals then the grammar is said to be an operator grammar.



For More Study Materials : www.keralanotes.com

**3.12 Definition**

Given an arbitrary CFG then the parsing of a string can be done in exponential time with back up.

**3.13 Definition**

Parsing techniques are classified as top down parsing techniques (or) bottom up parsing techniques depending on the way the syntax tree is built.

**3.14 Definition**

Top down parsing algorithms build the derivation tree top down, starting from the root and working down to the leaves. Top down parsing algorithms simulate a leftmost derivation.

**3.15 Definition**

Bottom up parsing algorithms build the derivation tree bottom up, starting from the leaves & working to the root. Bottom up parsing algorithm simulate the reverse of a rightmost derivation.

**3.16 Concept**

An arbitrary CFL suffices from three difficulties when constructing a parser algorithm.

1. Non-determinism – choice of right parts of rules
2. Left recursion
3. Ambiguity – more than one derivation tree for some sentence.

**3.17 Theorem:**

It is un-decidable if a CFG is ambiguous.

**3.18 Definition**

Practical parsing algorithm must parse the input in linear time & to do this are empty deterministic context free languages (or a subset of them)

**3.19 Definition**

Practical parsing algorithm use a deterministic push down automata as the parser.

keralanotes.com



For More Study Materials : www.keralanotes.com

# TOP DOWN PARSING

## 3.20 Introduction

We will consider the language of expression in C with +&*. The following are expressions.

i + i

i + i * i

(i + i) * i

There are an infinite number of valid expressions in C. Each expression is finite. We need a finite description for this infinite set of expression. We will use a context free grammer ( CFG)

## 3.21 Attempt I:

$E1 \rightarrow E\ \#$

$E \rightarrow E + E$

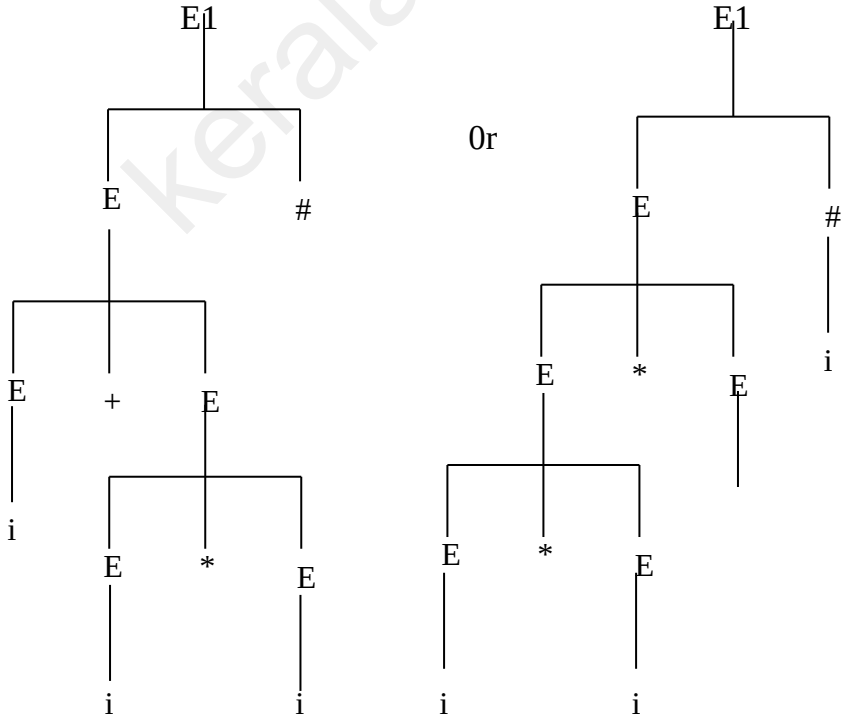
/E * E

/ (E)

/ i

The above is called a context free gammar. The meta symbols or non –terminals are {E1, E} the terminal set is [ i, + , *, (, ) #}. (Assume # is used to symbol the end of the input).

Now let us consider a sample expression i + i * i #. The expression have the two following derivation trees.




keralanotes.com



For More Study Materials : www.keralanotes.com

As there are two distinct derivation trees for the sentence i + i * i, the grammar is ambiguous. We can represent the derivation by two leftmost derivations.

E1⇒E #⇒ E+ E#⇒ i + E #

⇒ i + E* #⇒ i + i * E #

⇒ i + i * i #

Or

E1⇒E #⇒ E*E#⇒ E* E #

⇒ i + E* #⇒ i + i * E #

⇒ i + i * i #

There are two distinct leftmost derivations. The grammar is ambiguous.

We can construct rightmost derivations.

E1⇒E #⇒ E+ E#⇒ E+ E * E#

⇒ E + E* #⇒ E+ i * i #

⇒ i + i * i #

Or

E1⇒E #⇒ E*E#

⇒ E*i #⇒ E+E* i #

⇒ E + i * i #⇒ i + i *i #

**3.22 Strategy:**

First the ambiguity has to be removed. With the ambiguity it is not known as to whether + has a higher precedence that * or the way around.

By trial & error, the following grammar has been answered for the string i+ i*i.

E1 → E #

E→ E + T/T

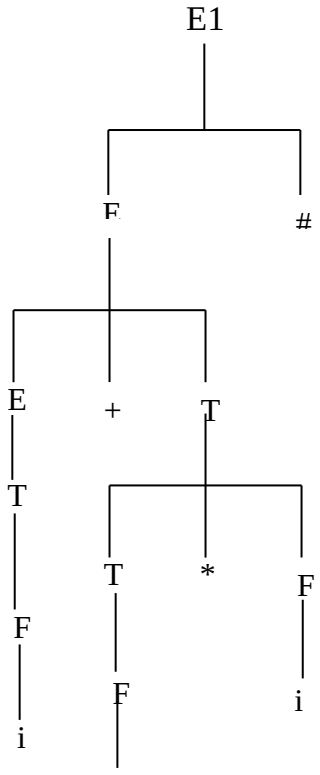
T → T * F/F

F → (E) / i



For More Study Materials : www.keralanotes.com

If we consider sentence i + I * I # there is a unique derivation tree for the sentence.




In the above grammar '*' occur lower than '+'. In a bottom up reduction the operator * is handled first & then the operator +. This means * has a higher priority than +.

**3.23 Theorem:**

This grammar can be shown to be unambiguous

**Problem:**

The grammar E1 → E #

E →E + E | E* E | (E) | a is left recursive. If we try to build a derivation tree top down, we may end up in the tree.

keralanotes.com



For More Study Materials : www.keralanotes.com

The infinite tree does not terminate. Top down parsing algorithms do not tolerate left recursive.

**3.24 Theorem:**

Any context free grammar can be put into the Griebach Normal Form (GNF) which is free from left recursive.

**Example 3.1:**

E1 E #

E E + E | E * E | (E) | i

Convert to the Griebach Normal Form

**Sole:** E1 → E #

E → i Z | (E) Z

Z→ +E Z | * E Z

**Example 3.2:**

E1 E #

E E + T / T

T T * F/ F

F ( E) | i

Convert to avoid left recursion.

**Sole:** E1 → E #

E → T + E | T

T → F * T/ F

F → (E) | i

**3.25 Concept**

Recursion can be replace by iteration.

## LL (I) PARSERS

- LL (I) is most widely used Top down parser
- LL (I) parser parse the given input string with help of LL (I) parsing table
- Given a grammar, two functions First ( ) and Follow ( ) are used to construct LL (I) parsing table.
  **First (α): Gives** set of terminals that begin in strings derived from α
- First (α) is computed using 3 rules



For More Study Materials : www.keralanotes.com

1. If $\alpha$ is terminal, First ($\alpha$) = {$\alpha$}
2. If $\alpha$ is non –terminal

a) and has $\varepsilon$ rule like $\alpha \rightarrow \varepsilon$, then First ($\alpha$ ) = {$\varepsilon$}

b) and has non null production like $\alpha \rightarrow X_1X_2X_3$ then

First ($\alpha$) = First ($X_1X_2X_3$ )

= First ($X_1$) if $X_1$ does not derive $\varepsilon$

= First ($X_1$) – { $\varepsilon$} $\cup$ First ($X_2X_3$) if $\Rightarrow\varepsilon$

Follow (A) : Gives set of terminals that may immediately follow to the right of A in any sentential form

- Follow (A) is computed using 3 rules
  1. If A is start symbol of G,
     Follow (A) = {$}
  2. If $X \rightarrow \alpha A\beta$ as in G,
     Follow (A) = First ($\beta$) - {$\varepsilon$}
  3. If $X \rightarrow \alpha$ A or $X \rightarrow \alpha A\beta$ {$\beta \Rightarrow \varepsilon$} is in G,
     Then follow (A) = follow (X)

**LL(I) parsing table construction:**

For each rule $A \rightarrow \alpha$ in G, apply following steps:

1. For each terminal a in First ($\alpha$), Add $A \rightarrow \alpha$ to M [A, a] where M is parsing table.
2. If first ($\alpha$), contains $\varepsilon$, add $A \rightarrow \alpha$ to M [A, b] for each b in follow (A)

- To construct an LL(I) parser, there is a restriction on grammar. The grammar has to be left factored and should be free of left recursion.
- Given a grammar, construct an LL(I) parsing table using the above procedure, If the LL(I) table has no multiple entries, then the grammar is called LL(I)
- Left recursive grammar cannot be LL(I)

keralanotes.com



For More Study Materials : www.keralanotes.com

**Example 3.3:**

Consider the following grammar. Compute FIRST & FOLLOW sets fore ach non – terminal. Check whether the grammar is LL(I)

S→ACB / CbB/Ba

A→da /BC; B→ g/ε; C→ h/ε

**Sol:** FOLLOW (B) = { a}∪ {#} = { a, #}

FIRST (B) = {g} ∪ FOLLOW (B) = {g,a,ε}

FIRST (C) = {h} ∪ FOLLOW (C)

= {h} ∪ {b}∪ FOLLOW (C) }

= { h, b, g, a, ε}

FIRST (A) = {d} ∪ { FIRST (B) }

= { d, g, a, ε}

FOLLOW (A) = FIRST (C) = { h,b, g, a, ε}

FIRST (C) = {b} ∪ FIRST (B)

= { g, a,b, ε}

Let us construct the predictive parsing table for the grammar

| | a | b | d | g | h | $ |
|---|---|---|---|---|---|---|
| S | S→ACB | | S→ACB | S→ACB | | S→ACB |
| A | | S→CbB | | S→CbB | S→CbB | S→CbB |
| B | | | | | | |
| C | | | | | | |
| A | | | | | | |

So the grammar is not LL(I)

**Example 3.4:**

Computer FIRST and FOLLOW for each non-terminal in the grammar,. Check if the grammar is LL(I)

S→aABb, A→ c/ε; B→ d/ε



For More Study Materials : www.keralanotes.com

**Sol:** FIRST (S) = A

FOLLOW 9S) = $

FIRST (B) = {d} ∪ FOLLOW (b)

= { d,b }

FOLLOW (B) = {b}

FIRST (A) = {c} ∪ FOLLOW (A)

= {c,b}

FIRST (A) = {b}

The predictive parsing table, so the grammar is LL (I)

| | a | b | d | d | $ |
|---|---|---|---|---|---|
| S | S→aABb | | | | |
| A | | A→ε | A→c | | |
| B | | B→ε | | B→d | |

**Example 3.5:**

**Construct the LL(i) parsing table and verify whether it is LL(I) or not**

S→aBDh

B→ cC

C→bC/ε

D →EF

E→g/ε

F→f/ε

**Sol:** FIRST (S) = { a}, FIRST (B)= c

FIRST (F) = {f} ∪ FOLLOW (F)

= {f} ∪ FOLLOW (D)

= {f, h}

keralanotes.com



For More Study Materials : www.keralanotes.com

FIRST (E) = {g} ∪ FOLLOW (E)

= {g} ∪ FIRST (F)

= { f,g,h}

FIRST (C) = {b} ∪ FOLLOW (C)

= {b} ∪ FOLLOW (B)

= {b} ∪ FIRST (D)

= {b} ∪ FIRST (E)

= { b, f,g, h}

FIRST (D) = FIRST (E) = { f,g, h}

The predictive parsing table

| | a | b | c | f | g | h | $ |
|---|---|---|---|---|---|---|---|
| S | S→aBDh | -- | -- | | | | |
| B | | | B→c | | | | |
| D | | C→bC | | C→ε | E→ε | C→ε | |
| D | | | | D→EF | D→EF | D→EF | |
| E | | | | E→ε | E→g | E→ε | |
| F | | | | F→f | | F→ε | |

So the grammar is indeed LL(I)

**Example 3.6:**

**Construct the LL(I) parsing table and verify whether it is LL(I) or not**

S→AaAb/BbBa

A→ ε

B→ε

**Sol:** FIRST (A) = FOLLOW (A)

= { a, b}

FIRST (B) = FOLLOW (B)

= { a, b}



For More Study Materials : www.keralanotes.com

FIRST (S) = FIRST (A) ∪ FIRST (b)

= { a, b}

The Predictive parsing table

**Example 3.7:**

**Construct the LL(I) parsing table and verify whether it is LL(I)**

S→1AB/ε

A→ IAC/OC

B→OS

C→1

**Sol:** FOLLOW (S) = $

FIRST (S) = {I, $}

FIRST (A) = {0,1}

FIRST (B) = {0}

FIRST (C) = {1}

The predictive parsing table

| | 0 | 1 | $ |
|---|---|---|---|
| S | | S→1AB | S→ε |
| A | A→ OC | A→1AC | |
| B | B→0S | | |
| C | | C→1 | |

The grammar is LL(I)

**Example 3.8:**

**Check whether the following grammar is LL (I)**

S→A

A→ aB/Ad

B→bBC/f

C→g

keralanotes.com



For More Study Materials : www.keralanotes.com

**Sol:** The left recursive rule A → Ad makes the grammar unsuitable for top down parsing. So the grammar is not LL (I)

**Example 3.9:**

**Convert the following grammar to an unambiguous grammar. Check whether the resulting grammar LL(I)**

R→R + R/R R/ R* /(R) / a/b

**Sol:** We will have the order of precedence +,--, # An unambiguous grammar is

R→R + T/ T

T → TF/F

F → F*/(R)/a/b

Factoring yields

R→ T (+T)

T → F (F)

F → F*/(R)/a/b

Converting the BNF to standard CFLs

R→TA

A → ε/+TA

T → FB

B → ε/FB

F →F */(R)/a/b

Removing the left recursive F – rule

R→TA

A → ε/+TA

T → FB

B → ε/FB

F → (R) Z/aZ/bZ

Z → ε/*Z

keralanotes.com



For More Study Materials : www.keralanotes.com

Computing the FIRST and FOLLOW sets

FIRST (F) = { (,a,b}

FIRST (T) = FIRST (F) = { (,a,b}

FIRST (R) = FIRST (T) ={ (,a,b}

FIRST (A) = {+} ∪ FOLLOW (A)

= {+, #, ) }

FIRST (B) = FIRST (F) ∪ FOLLOW (B)

= FIRST (F) ∪ FOLLOW (T)

= FIRST (F) ∪ FIRST (A)

= { (, A, B, +, #) }

The predictive parsing table is

The above is LL (I)

**Example 3.10:**

**Check whether the following grammar is LL (I)**

S→ I E t SSI / a

SI→ε/ eS

E →b

**Sole :** FIRST (E) = b

FIRST (S)= { i,a }

FIRST (SI)= { e}∪FOLLOW (SI) = {e, #, i}

The predictive parsing table is

| | i | t | e | a | b | # |
|---|---|---|---|---|---|---|
| S | S→iEiSSI | | | S→a | | |
| SI | | SI→Es | | | | SI→ε |
| | | SI→ε | | | | |
| E | | | | E→b | | |

The grammar is not LL (I)



For More Study Materials : www.keralanotes.com

**Example 3.11:**
**Check whether the following grammar is LL (I)**

E→ aE / (E )

A→ + E / *E/ε

**Sole :** FIRST (E) = { a, ( }

FIRST (A)= {+, * } ∪FOLLOW (A)

= {+, * #, )}

The predictive parsing table is

| | a | + | * | ( | ) | # |
|---|---|---|---|---|---|---|
| E | E→ aA | | | E → (E) | | |
| A | | A → +E | A→ * E | | A →ε | A → ε |

The grammar is LL (I) as there is no conflict in the table.

**Example 3.12:**

**Check whether the following grammar is LL (I)**

SI→ S#

S→ aAa/ε

A→ abS/c

**Sole :** FOLLOWS (S) = { # }
FIRST (S)= {a } ∪FOLLOW (S)
= {a, # }

The predictive parsing table follows

| | a | b | c | # |
|---|---|---|---|---|
| S1 | S1 → S# | | | S1→ S# |
| S | S → aAa | | | S→ ε |
| A | A→abS | | A→ c | |

The grammar is LL (I)



For More Study Materials : www.keralanotes.com

**Example 3.13:**

**Check whether the following grammar is LL (I)**

S→ A#

A→ Bb/Cd

B→ aB/ε

C→ cC/ε

**Sole :** FOLLOWS (B) = { b }
FOLLOW(C)= {d }
= {a, # }

The predictive parsing table is

| | a | b | c | d | # |
|---|---|---|---|---|---|
| S | S → A# | S → A# | S → A# | S → A# | |
| A | A→ Bb | A→ Bb | A→ Cd | A→ Cd | |
| B | B→ aB | B→ ε | | | |
| C | | | C→ cC | C→ ε | |

The grammar is LL (I)

**Example 3.14:**
**Can a LL(I) parser be constructed for the following grammar**

S→ xSx / Tx

T→ Tx / x

**Sole :** The rule T→ Tx / x is left recursive, so a LL (1) parser cannot be constructated for the grammar.

**Example 3.15:**
**Can a LL(I) parser be constructed for the following grammar**

S→ (S)

S→ SS / ( )

**Sole :** The grammar is left recursive and so is unsuitable for LL(I) parsing technique.



For More Study Materials : www.keralanotes.com

## Dangling else problem

**3.26.** Consider a HLL which allows both the if & if else statement

<statement > : : = if <expression > then< statement > | if < expression > then

<statement > else < statement >

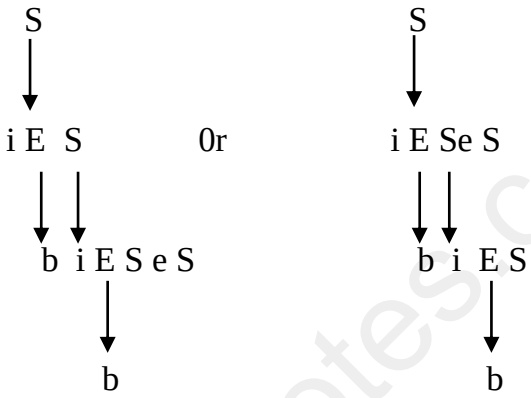
<statement > : = a

The above can be encoded in C as

S→ i E S | I E Se S| a

E→ b

Consider the construct i E I E Se S. This has two derivation trees in the grammar.




This is called the dangling else problem

The grammar is not LL(I) as there are two choices for S

**3.27.** S→ iESS$^1$

S$^1$→ eS/ε

E→ b

- FIRST & FOLLOW Sets

FIRST (S) = { i }

FIRST (E) = { b }

FIRST (S$^1$) = {e}∪ FOLLOWS (S$^1$)

= {e} ∪ {#}

= { e, #}\

keralanotes.com



For More Study Materials : www.keralanotes.com

- The predictive parsing table

| | i | e | b | # |
|---|---|---|---|---|
| S | $S \rightarrow iESS$ | | | |
| $S^1$ | | $S^1 \rightarrow eS$<br>$S^1 \rightarrow \varepsilon$ | | $S^1 \rightarrow \varepsilon$ |
| E | Conflict | | $E \rightarrow b$ | |

## Bottom Up Parsers

- Bottom up parsers are also known as SR parsers.
- There are two types of Bottom up parsers.
    1. Operator precedence parsers
    2. LR parsers.

## Operator Precedence Parsing

- **Operator precedence parser :** To design an operator precedence parser, the grammar
- **Operator grammar:** A grammar that does not contain any ε-rules and does not contain two adjacent non-terminals on right side of any production is called operator grammar

  Ex: $E \rightarrow E\ AE/\ id;\ A \rightarrow +/*/-//$ is not operator grammar equivalent operator grammar is

  **Ex:** $E \rightarrow E+E|\ E*E|\ E-E|\ E/E|\ id$
- **Parsing algorithm:** If a is top of stack & b is symbol pointed by input pointer.

  **1.** If a = b = $ successful completion of parsing

  2. If a < b or a = b then push b onto stack and advance input pointer to next symbol.

  3. if a > then repeat pop stack until the top of stack is related by < to the terminal most recently popped.

keralanotes.com



For More Study Materials : www.keralanotes.com

**3.28** An operator grammar should not have ε productions. An operator grammar should not have two adjacent non-terminals in the right hand side of any rule.

**3.29** Operator grammar for expressions

$$E \rightarrow E+E \mid E*E \mid (E) \mid id$$

**3.30 Operator Precedence Table**

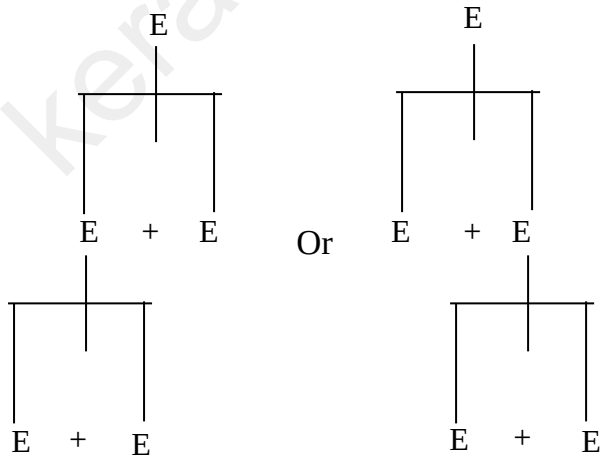
Though an operator grammar may be ambiguous, a deterministic shift, reduce parser can be constructed for it. Conflicts resulting in ambiguity are resolved using the Associativity and precedence of operators.

**3.31** In the reverse of a rightmost derivation, the string (sentential form) that is replaced by the left hand side of the rule is called the handle.

3.32 Construction

| | + | * | ( | ) | Op | # |
|---|---|---|---|---|---|---|
| + | | | | | | |
| * | | | | | | |
| ( | | | | | | |
| ) | | | | | | |
| id | | | | | | |

- E⇒ E+ E ⇒ E + E +E



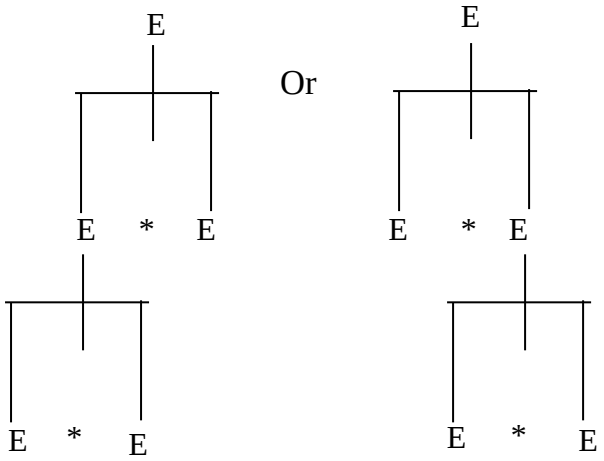

The rule of a language like C say + is left associative. The first tree should be chooses.

So, + .> +



For More Study Materials : www.keralanotes.com

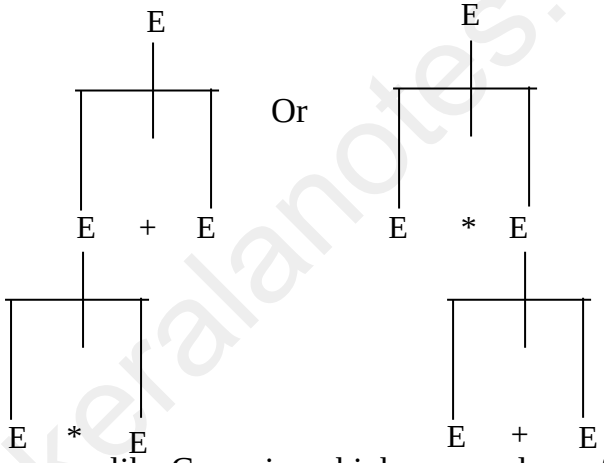
- E⇒ E+ E ⇒ E * E *E




The rule of a language like C demand that * be left associative. So the second of the trees invalid. So, # .> *

- E⇒ E+ E ⇒ E + E *E or E⇒ E* E ⇒ E + E *E

The above derivations yield the derivation trees.

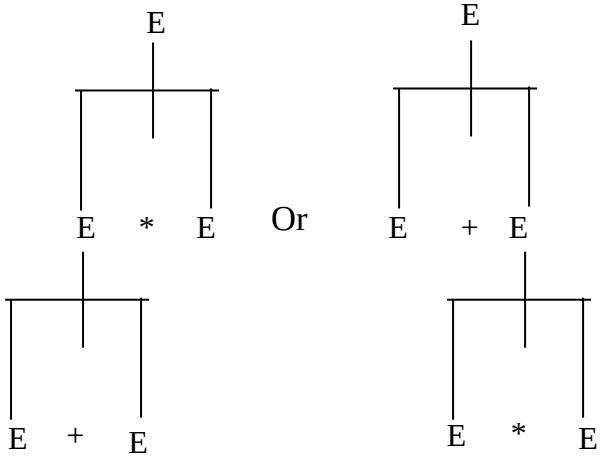



The rules of a language like C require a higher precedence for * than +, Sot eh second tree valid.

So, +< . *

- E⇒ E* E ⇒ E * E +E or E⇒ E+ E ⇒ E * E +E

The above derivations yield the derivation trees.



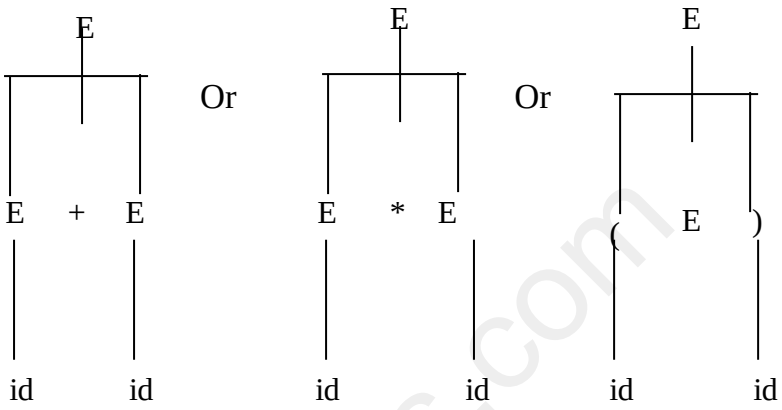



For More Study Materials : www.keralanotes.com

The rules of a language like C require a higher precedence for * than +, Sot eh second tree valid.

So, *. >+

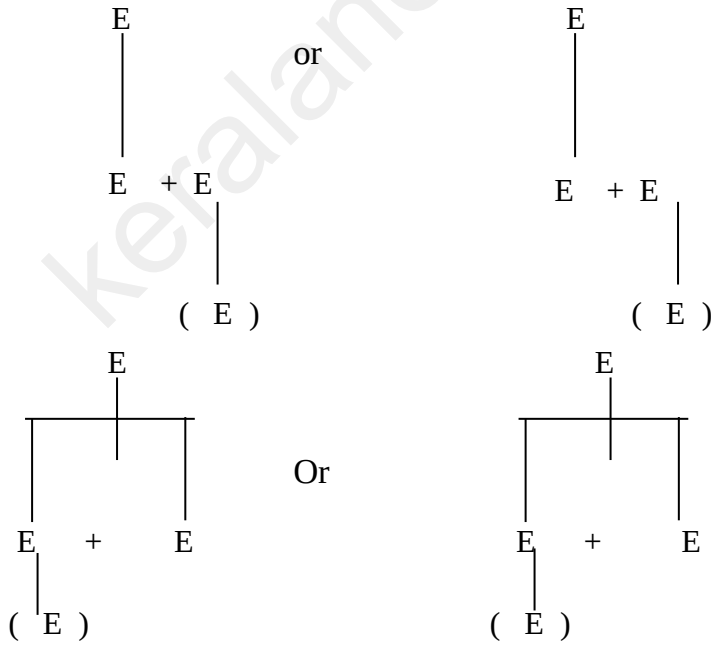
- The rules of a language like C do not allow the string idi, id (,
  So id no relation id, id no relation (
- Also there is only one end of file symbol.
  So, # no relation #
- The trees




Yield the relations

id .> + , id .> *, id .> )

- Consider the trees




The above trees yield the relations

+ <. (, * <. (, ) .> +, ) .> *



For More Study Materials : www.keralanotes.com

- The operator precedence table

| | id | + | * | ( | ) | # |
|---|---|---|---|---|---|---|
| id | - | .> | .> | - | - | .> |
| + | < . | .> | < . | < . | .> | .> |
| * | < . | .> | .> | < . | .> | .> |
| ( | < . | < . | < . | < . | =0 | - |
| ) | - | .> | .> | - | .> | .> |
| # | < . | < . | < . | < . | - | - |

**Example 3.16:**
**Convert the following grammar to an operator grammar**

S→ SAS/a

A→ bSb/b

**Sole :** Substitution the A productions

S→ SbSbS/SbS/a

A→ bSb/b

The above grammar is an operator grammar.

**Example 3.17:**
**Convert the following grammar to an operator grammar**

P→ SR/S

R→ bSR/bS

S→ WbS/W

W→ L*W | L

L→ id

**Sole:** Only the rule P →SR violates the operator grammar form. Substituting the R-Production.

P → SbSR/SbS/S

R → bSR/bS

S → WbS/W

W → L * W/ L

L → id

keralanotes.com



For More Study Materials : www.keralanotes.com

Though not essential we can eliminate the unit productions

P → SbSR/SbS/S

R → bSR/bS

S → WbS/L*W/L

W → L * W/ id

L → id

Or

P → SbSR/SbS/S

R → bSR/bS

S → WbS/L* W/id

W →L * W/id

L → id

Or

P → SbSR/SbS/WbS / L*W/id

R → bSR/bS

S → WbS/L* W/id

W →L * W/id

L → id

**Example 3.18:**
**Prepare the precedence table and parse tree for id * id b id* id S**

**Slo:** P→ SbSR/SbS/R

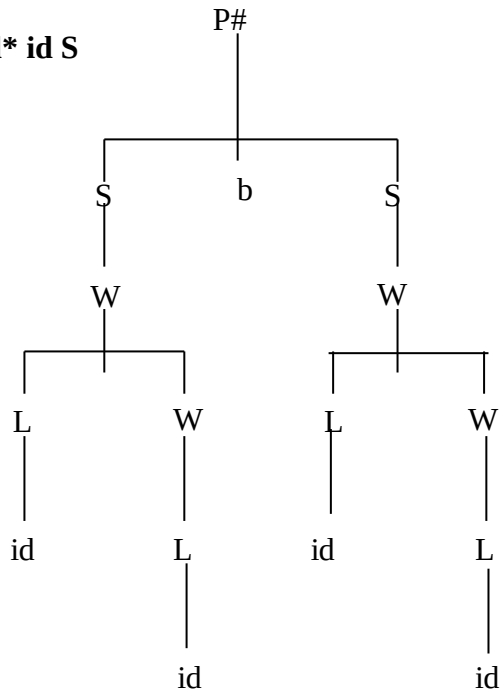
R→ bSR/bS

S→ WbS/W

W→ L*W | L

L → id

The parse tree



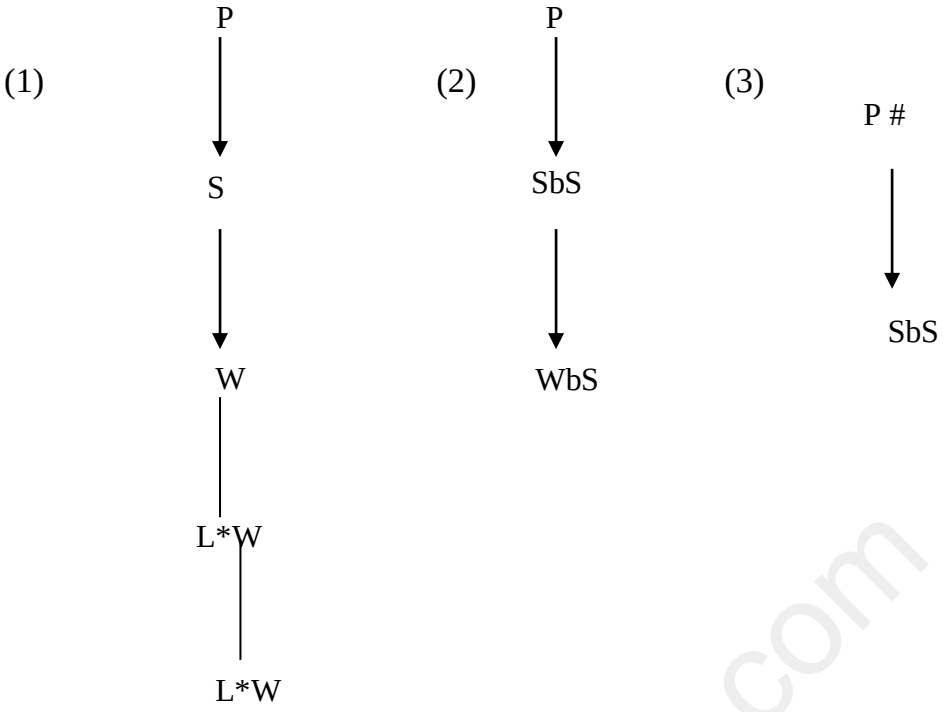

The precedence table

| | b | * | Id | # |
|---|---|---|---|---|
| b | .> | < . | < . | .> |
| * | .> | .> | < . | .> |
| Id | .> | .> | | .> |
| # | | | | |



For More Study Materials : www.keralanotes.com

To obtain the above precedence table we consider a number of syntax trees




## The LR Parsing Techniques

- **LR Parsers** : are of three types

  1. SLR (1)  2. LALR(1)  3. CLR (1)

- **SLR (1)** : Simple LR (I) parser is constructed with LR(O) items
- **LALR (I) & CR (I)** : Look ahead LR(I) and canonical LR (I) parsers are constructed with LR(I) items.
- **Parsing algorithm**: **If s** is state on top of stack and is a symbol pointed by input pointer, initially push 0 (initial state of DFA) on top of stack and advance input pointer to next symbol.

  1. if action [s,a] = $s_i$ push a then i onto stack and advance input pointer to next symbol.
  2. if action [s, a] = $r_i$ if $r_1$ is A $\rightarrow\beta$, pop off 2x / $\beta$/ symbol from stack replace by A. if $s^1$ is state below, a then push go to [$s^1$, A] on to stack. Output production A $\rightarrow\beta$,
  3. if action [s,a] = accept, successful completion
  4. if action [ s, a] = blank, error.



For More Study Materials : www.keralanotes.com

- **SLR (1) parsing table construction:**

  1. Create canonical collection of LR (0) items

  2. Draw DFA and convert to SLR (I) table

- To create canonical collection of LR (0) items, we require two functions closure () & Goto ()

  1. Create canonical collection of LR (0) items

  **Closure (I) : gives set of items by applying following two rules**

  1. Initially add every item from 1 to closure I
  2. if $A \rightarrow \alpha. B\beta$ is in I, then add $B \rightarrow r$ to closure I. Repeat this step for every newly added item until no more items can be added.

  **Ex: E** $\rightarrow$ E + T closure of ( $E^1 \rightarrow$. E) is

| | |
|---|---|
| /T | $I_0$ $E^1 \rightarrow$ .E |
| T$\rightarrow$ T*F | E$\rightarrow$ .E+T/.T |
| /F | T$\rightarrow$.T* F/.F |
| F$\rightarrow$id | F$\rightarrow$.id |

  **Goto** (I): Closure of set of all items $A \rightarrow \alpha$ x .$\beta$ such that $A \rightarrow \alpha$ x $\beta$ is in I.

  **Ex:** Goto ($I_0$, E) in above example is

| | |
|---|---|
| $E^i \rightarrow$E; | E$\rightarrow$E. + T |

- **Constructing SLR Parsing table:**

1. Construct canonical collection of set of items C = { $I_0$, $I_1$, ……$I_a$} for $G^1$.
2. State I in table is constructed from each $I_i$ in C. parsing actions for state I are given by.

   a) if goto ($I_i$ a) = $I_i$, then set action [ i, a] = $S_j$

   b) if goto ($I_i$ A) = $I_i$, then set action [ i, A] = j
3. If $A \rightarrow \alpha$ is in $I_i$, then set action (I, a) to reduce by $A \rightarrow \alpha$ for all a in follow (A)
4. if $S^1 \rightarrow$ S. is in $I_i$, then set action [ i, $] to accept.

- The same procedure is followed for constructing parsing tables LALR (I) & CLR (I) but with LR(I) items.
- The main difference between SLR (I) & CLR (I) lies in reduce entries.



For More Study Materials : www.keralanotes.com

- In LALR (I) & CLR (i) Tables whenever a state $I_i$ contains LR(I) item A →a, $/a/b
- LR(O) ⊆ SLR(I) ⊆ LALR(I) ⊆ CLR(I)
- Every LR(0) grammar is SLR (I)

**Example 3.19**

**Check whether the following grammar is LR(0)**

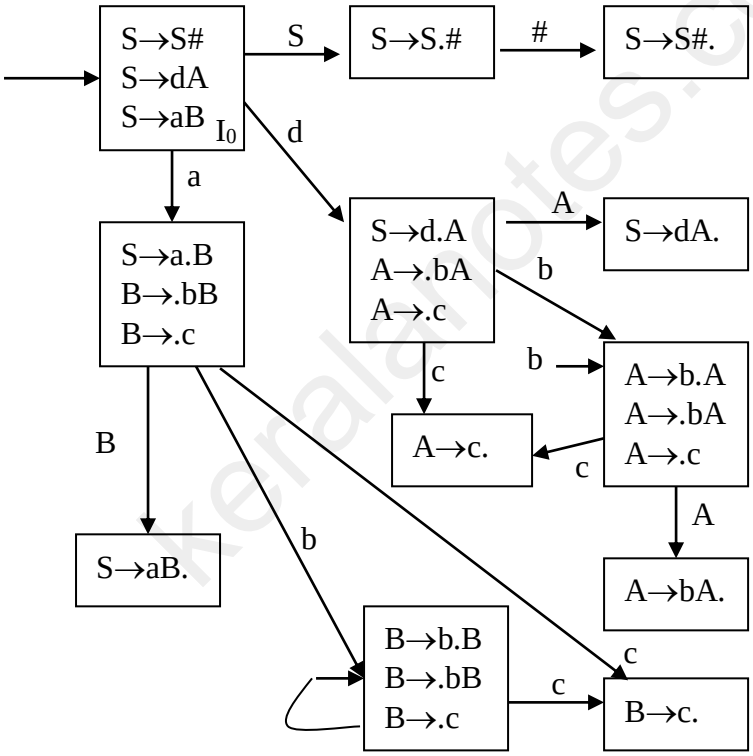
**S → S #**

**S → dA /aB**

**A→ bA/c**

**B→ bB/c**

**SoL:** The LR(0) machine is constructed




The grammar is LR(0)

b



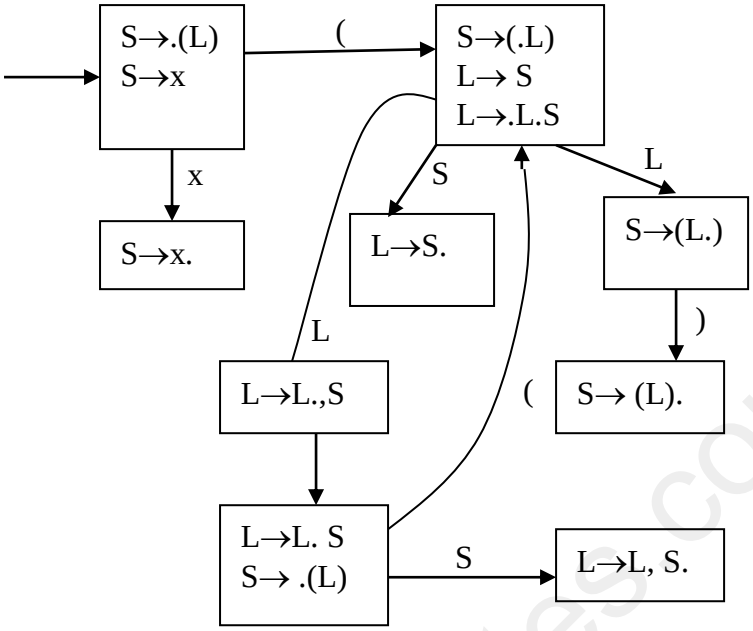
For More Study Materials : www.keralanotes.com

**Example 3.20**

**Check whether the following grammar is LR(0)**

**S → (L), S→ x, L → S, L → L, S**

Sol: The LR(0) machine follows




The grammar is LR(0)

**Example 3.21**

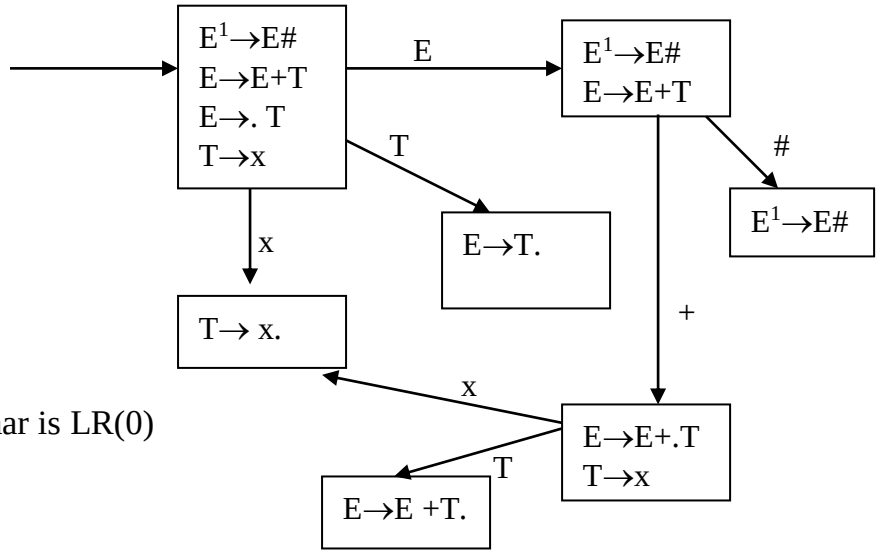
**Check whether the following grammar is LR(0)**

**E→ E #**

**E→ E + T| T**

**T→ x**

Sol: The LR(0) machine is constructed




The grammar is LR(0)



For More Study Materials : www.keralanotes.com

**Example 3.22**

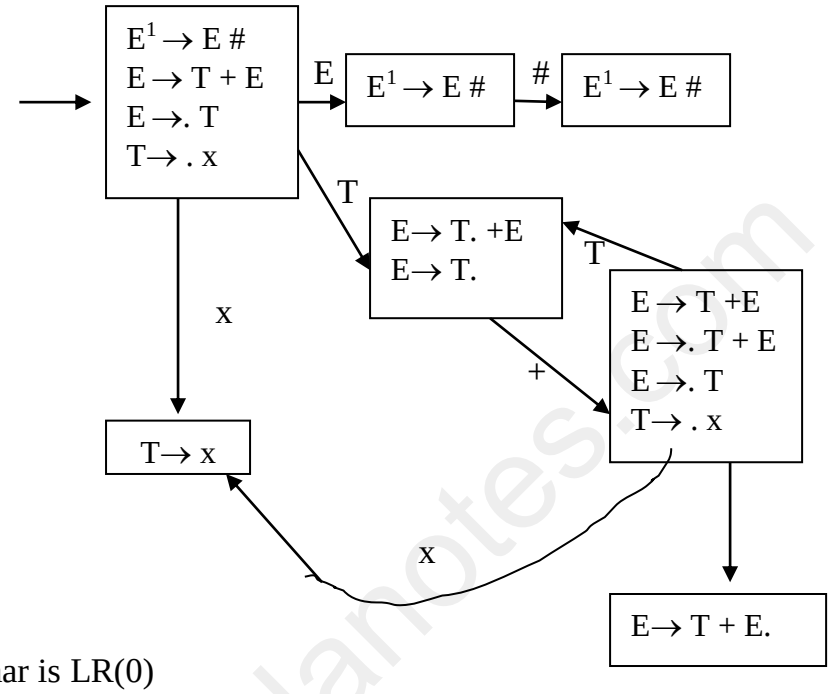
**Check whether the following grammar is LR(0), SLR(I)**

**$E^1 \rightarrow$ E #**

**E $\rightarrow$ T + E/T**

**T $\rightarrow$ x**

Sol: construct the LR(0) machine




The grammar is LR(0)

FOLLOW (e) = {#}

So we can use this to resolve the inadequate state. So the grammar is SLR (I)

**Example 3.23**

**Check whether the following grammar is LR(0), SLR(1)**

**S $\rightarrow$ A/ B**

**A $\rightarrow$ a A b/ab**

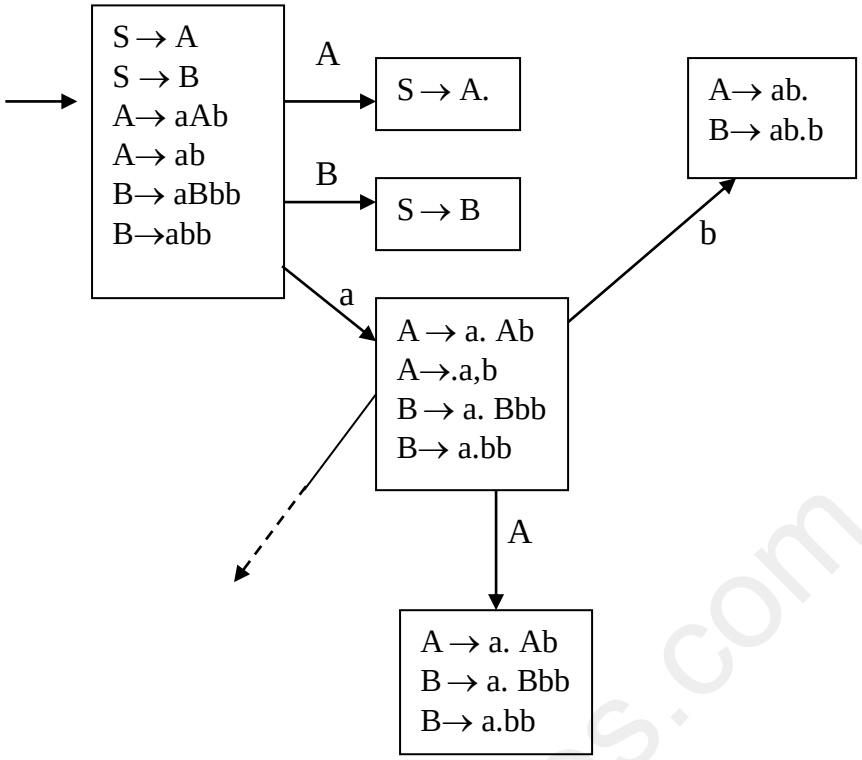
**B $\rightarrow$ a Bbb/abb**

keralanotes.com



For More Study Materials : www.keralanotes.com

Sol: Construct LR(0) machine




The grammar is not LR(0). As the incoming symbol 'b' as the FOLLOW symbol cannot resolve a conflict the grammar is not SLR (I)

**Example 3.24**

**Check whether the following grammar is LR(0), SLR(1)**

**E→ E + T / T**

**T → T * F/ F**

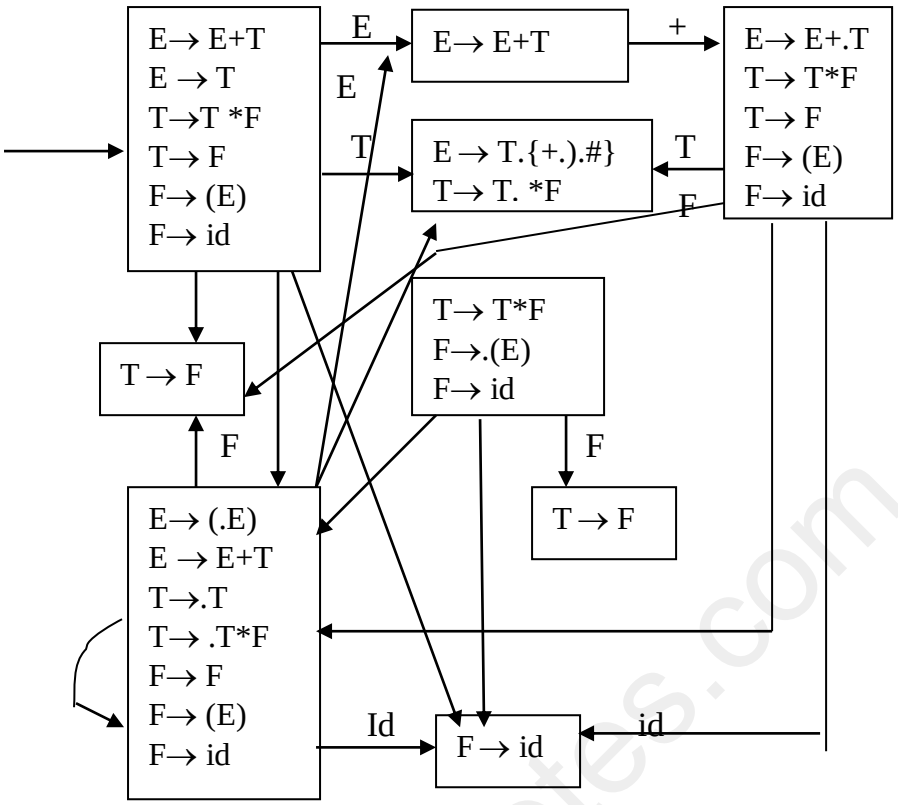
**F→ (E) / id**

keralanotes.com



For More Study Materials : www.keralanotes.com

Sol: Construct the SLR (I) machine




The grammar is SLR (i) & not LR (O)]

**Example 3.25**

**Check whether the following grammar is LR(0), SLR(1)**

**E→ E + T / T**

**T → T F/ F**

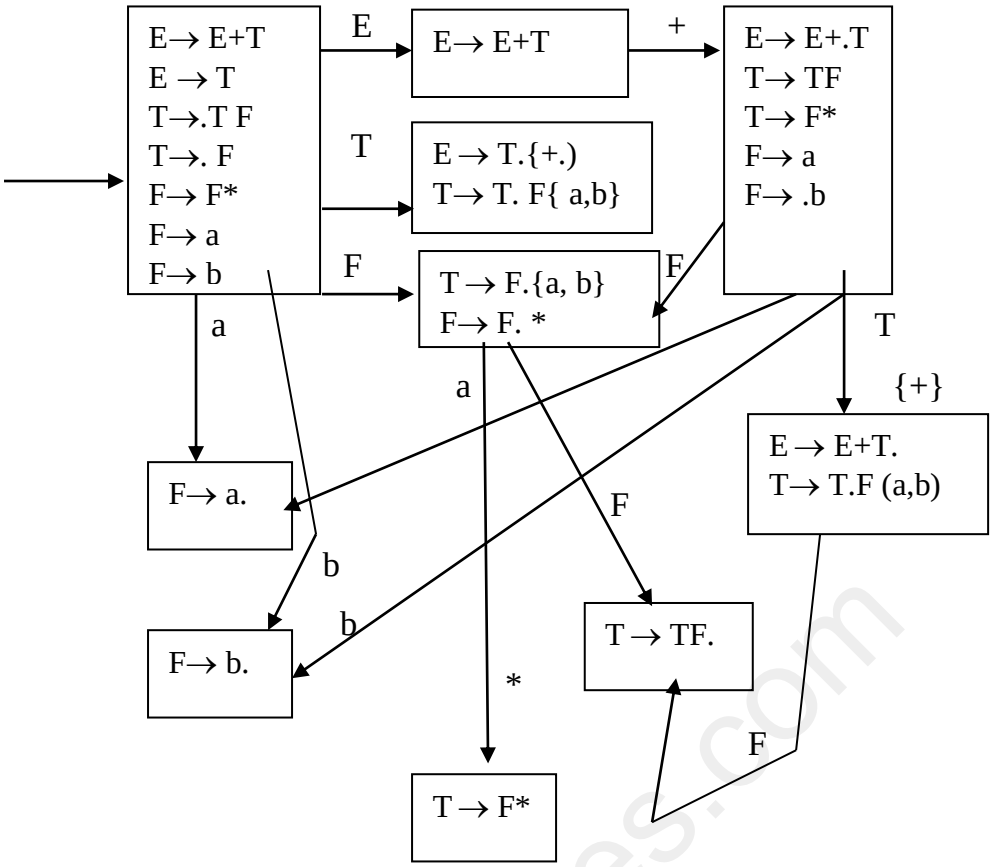
**F→ F * / a / b**

Sol: This is a grammar for regular expression which is SLR (i) & not LR (0) as shown earlier.

The LR (0) & SLR (I) machines follow



For More Study Materials : www.keralanotes.com




The grammar is not LR (0) but the inadequate states can be resolved by the FOLLOW symbols, Sot eh grammar is SLR (I)

**Example 3.26**

**Check whether the following grammar is SLR(1)**

**S → XY a#**

**X→ a/ Yb**

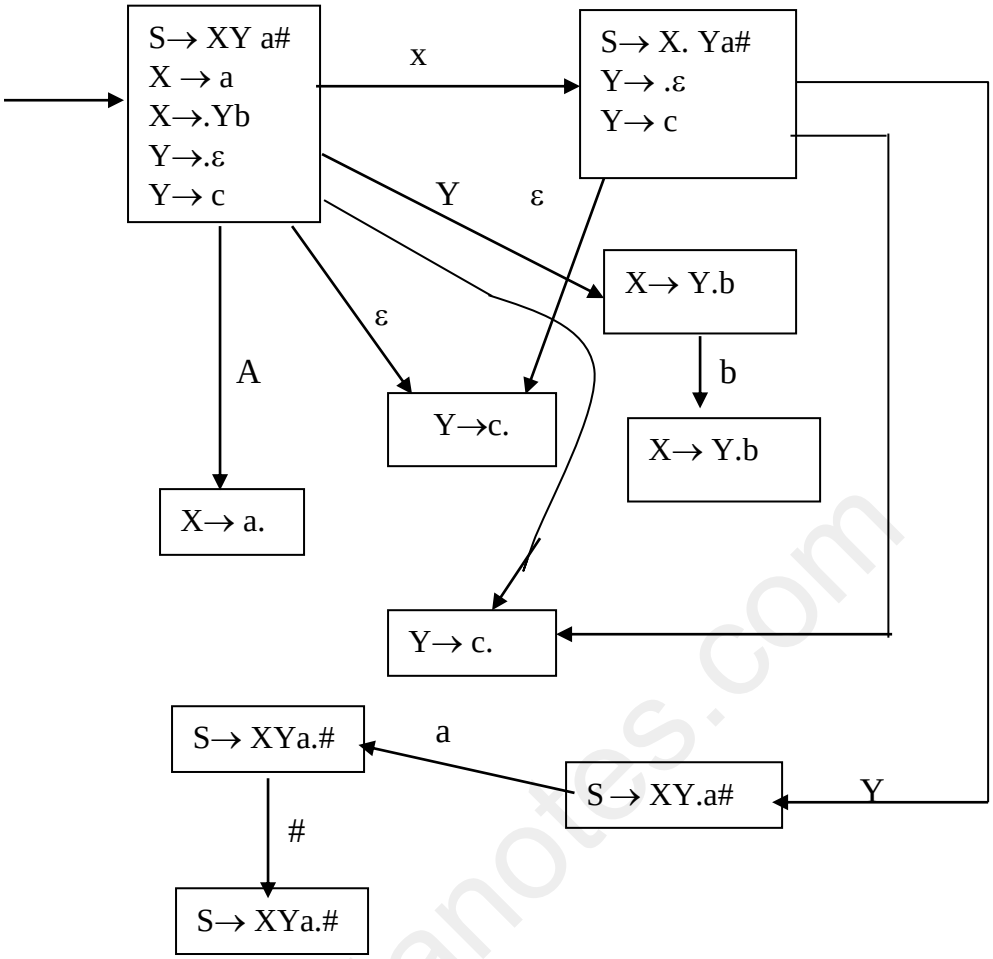
**Y→ ε /c**

keralanotes.com



For More Study Materials : www.keralanotes.com

**Sol: The LR (0) machine**




The grammar is SLR (1)

**Example 3.27**

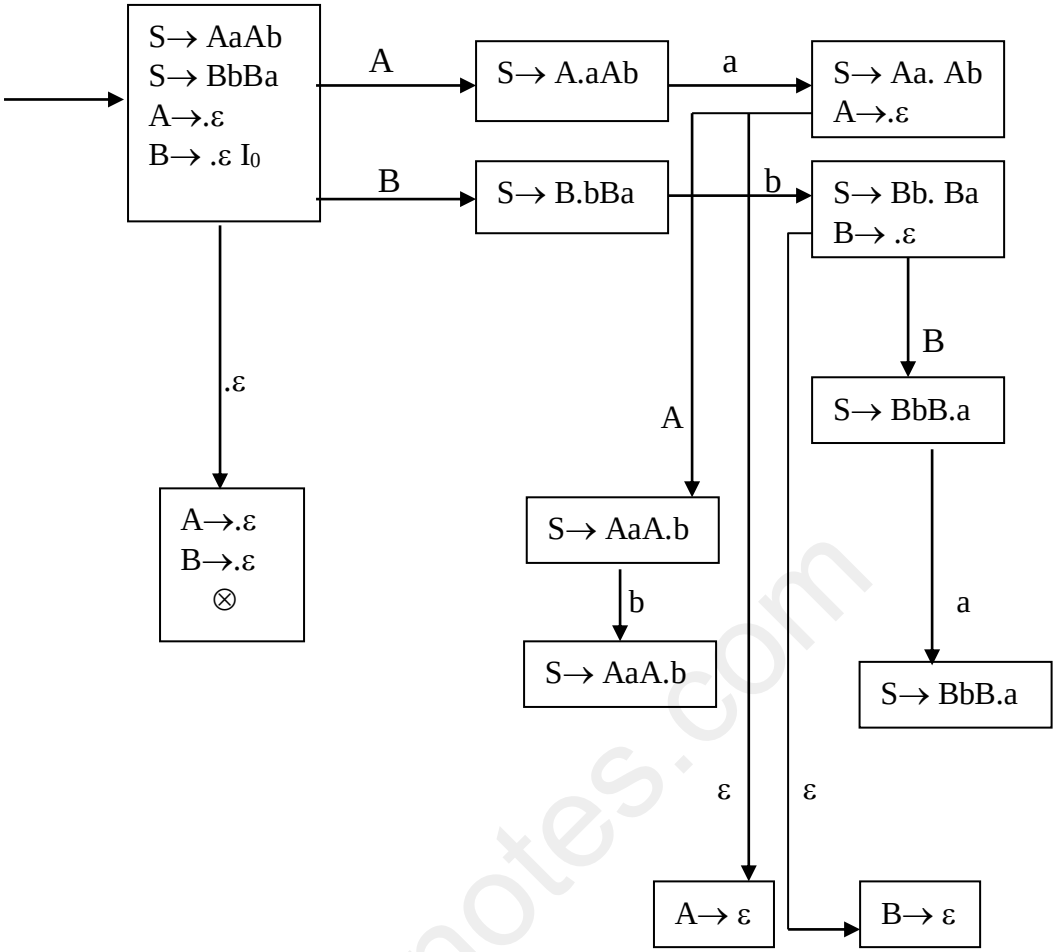
**Check whether the following grammar is LR (0), SLR (1), LALR(1) & LR (1)**

**S → aAaB, S→ BbBa, A→ ε . B → ε**

**Sol: the LR (0) machine**



For More Study Materials : www.keralanotes.com




The grammar is not LR (0) as ⊗ in an inadequate state.

FOLLOW (A) = {a, b}

FOLLOW (B) = {a, b}

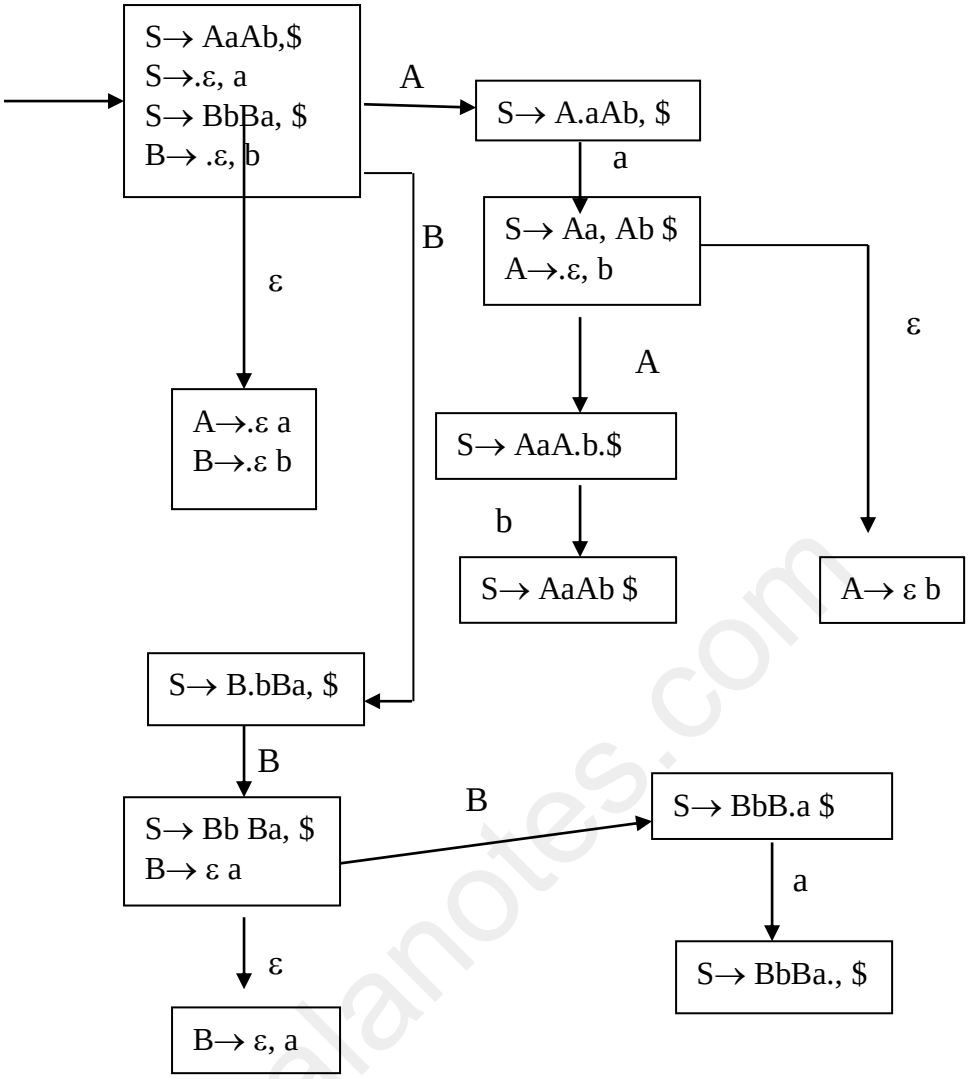
So we cannot resolve the state ⊗ based on the FOLLOW symbol. So the grammar is not LR (0)

keralanotes.com



For More Study Materials : www.keralanotes.com

The LR (1) machine




**Example 3.28**

**Check whether the following grammar is LR (0), SLR (1), LALR(1) & LR (1)**

**$S^1 \rightarrow S$ #**

**S→ Aa / b Ac / d/ bda**

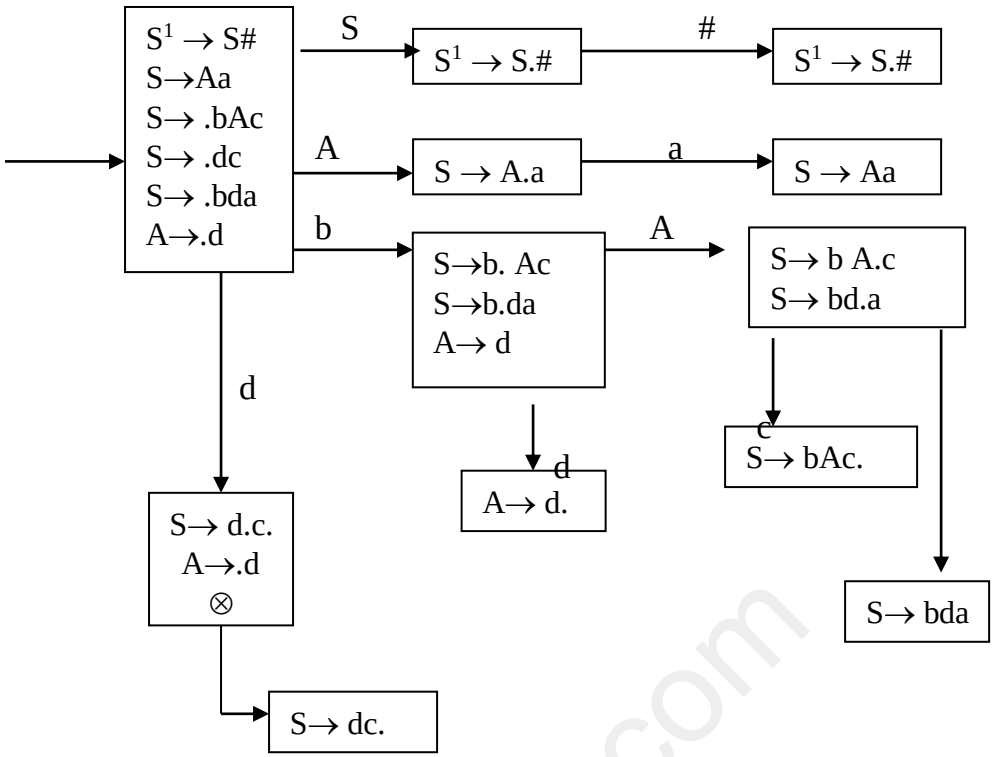
**A→ d**

**Sol: the LR (0) machine**

keralanotes.com



For More Study Materials : www.keralanotes.com




As there is a shit / reduce conflict in ⊗ the grammar is not LR (0)

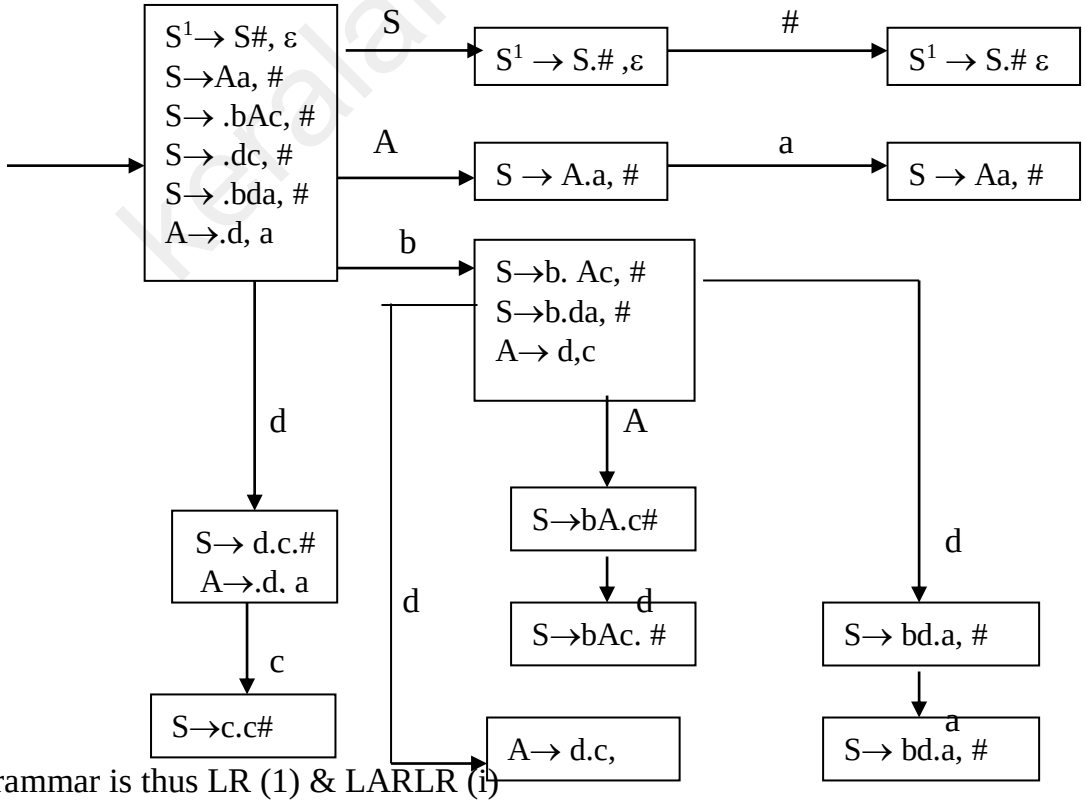
FOLLOW (S) = { #}

FOLLOW (A) = {a, c}

So we cannot resolve the conflict in state ⊗ based on the follow symbol.

So the grammar is not SLR (1)

The LRLR (1) machine for the grammar




The grammar is thus LR (1) & LARLR (1)



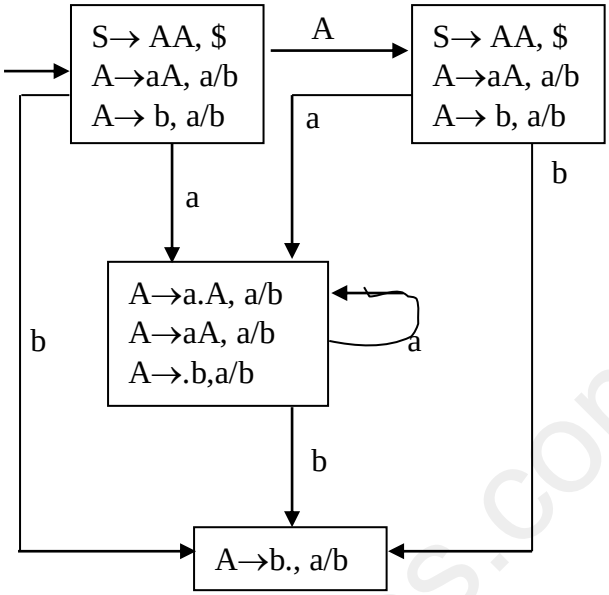
For More Study Materials : www.keralanotes.com

**Example 3.29**

**Check whether the following grammar is LALR(1) & LR (1)**

**S → AA, A →a A/b**

**Sol: the LR (1) machine**

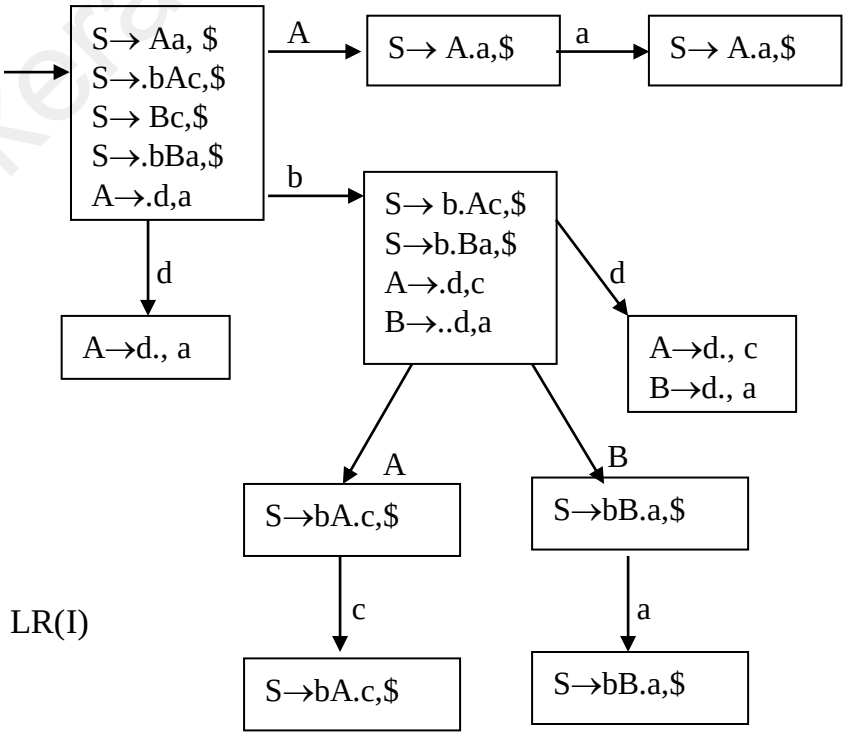



**Example 3.30**

**Check whether the following is LR (1)**

**S → Aa/b Ac/ Bc / b Ba**
**A → d**
**B → d**

**Sol:** the LR (1) machine for the grammar




The grammar is thus LR(I)



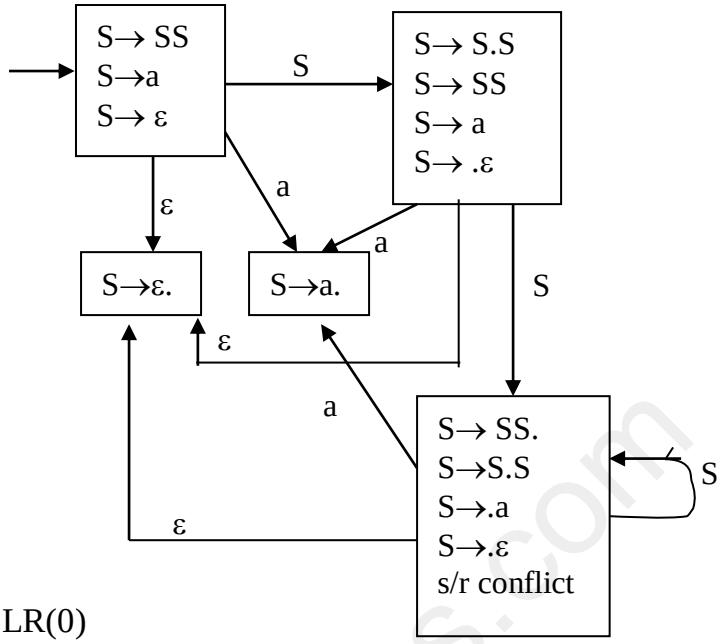
For More Study Materials : www.keralanotes.com

**Example 3.31**

**Check whether the following grammar is LR (0), SLR (I), LALR(1), LR(1)**

**S → SS/a/ε**

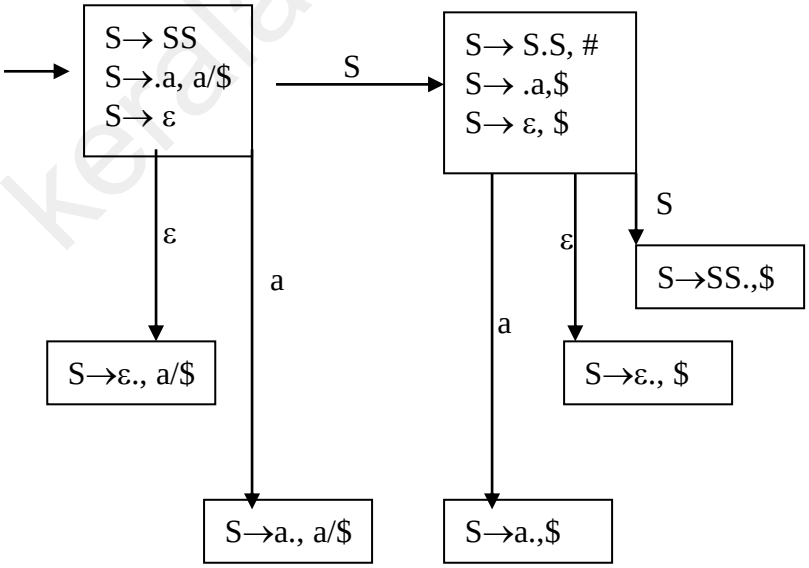
**Sol:** the LR (0) machine




The grammar is not LR(0)

FLOOW (S) is {a, #} & does not resolve the inadequate state so the grammar is not SLR (1)

The LR (1) machine




It is not clear whether to use S → a or S→ε as the look ahead symbols are the same. The grammar is neither LR (1) nor LALR (1). The grammar is ambiguous & no ambiguous grammar can be LR



For More Study Materials : www.keralanotes.com

**Example 3.32**

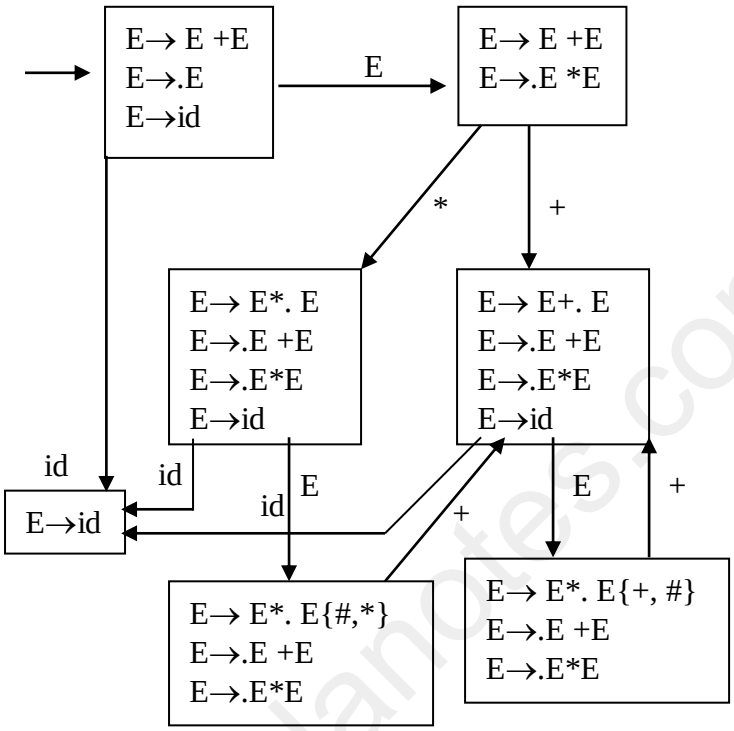
**Design SLR Parser**

**E → E + E**

**/E * E**

**/ id**

**Sol:** Construct the LR (0) machine & use follow symbols to resolve conflicts




ε

In the case of

E → E* E* E*…….

We use left associative of * and reduce.

In the case of E →. E* E + E…..

We use the higher precedence of * and reduce

In the case of E →.E+ E + E..

We use the left associativity of + and reduce

In the case of E→ E+ E*E…..

We use the higher precedence of * and shift.



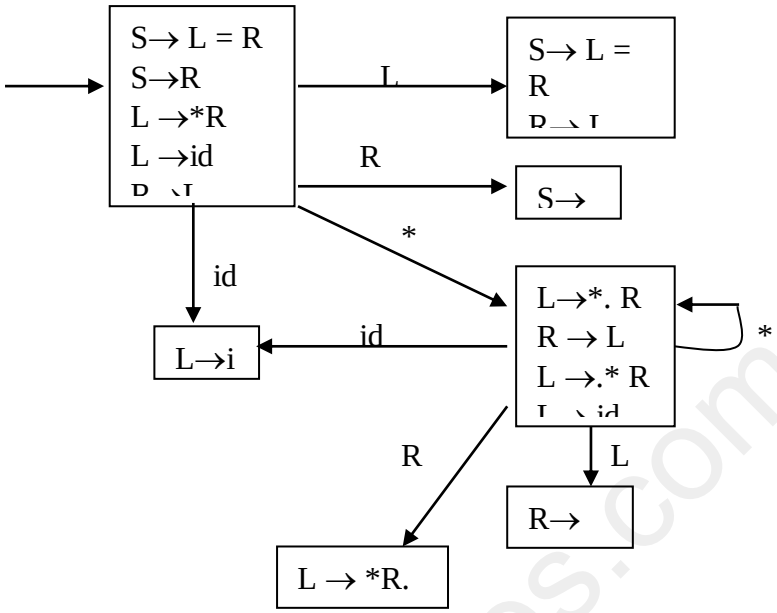
For More Study Materials : www.keralanotes.com

**Example 3.33**

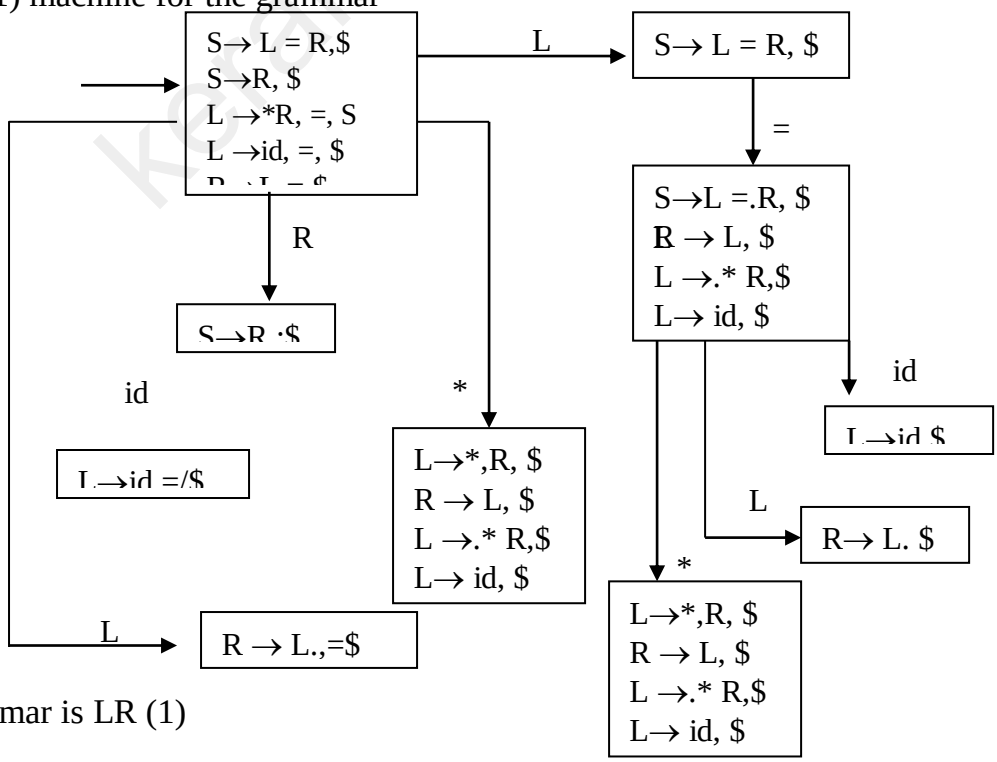
**Check if the following grammar is SLR (1) / LR (1)**

$$S \rightarrow l = R/R,\ L \rightarrow * R / id,\ R \rightarrow L$$

**Sol:** The LR (0) machine for the grammar




As there is s/r conflict in state ⊗, the grammar is not LR (0)
The FOLLOW (R)= { =, #}
So we cannot resolve the inadequate state ⊗ based on the FOLLOW Symbol. So the grammar is not SLR (1)
The LR (1) machine for the grammar




The grammar is LR (1)



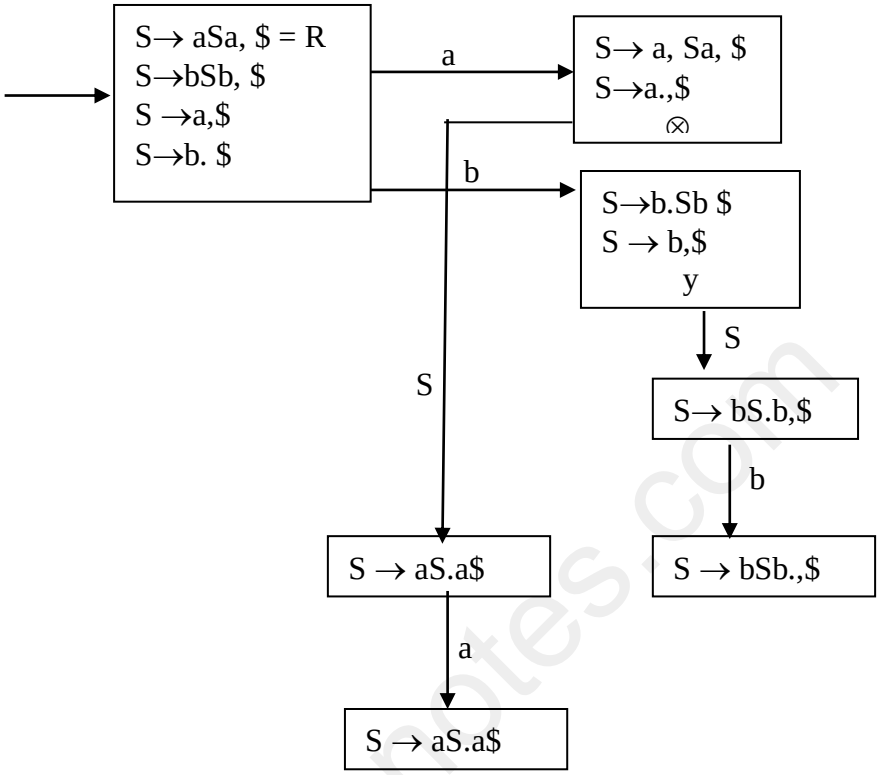
For More Study Materials : www.keralanotes.com

**Example 3.34**

**Check L(G) = {$ww^R \mid w \in \{a,b\}^*$ } is LR(1)**

**S → aSa/b Sb/a/b**

**Sol:** construct the LR (I) machine




The inadequate states ⊗ & ⓨ cannot be resolved by the lookahead set.

The grammar is not LR (1)

The language L = {$ww^R$ / w ∈ $(a+b)^*$ } is the set of all palindromes over {a,b}. It is a CFL which is not a DCFL. LR techniques are restricted to DCFLs.

**Example 3.35**

**How many conflicts occur in DFA with LR(1) items for the following grammar.**

**Sol:** There are 2 inadequate states in the LR(I) machine.

**Example 3.36**

**Find the closure of $E^1$ → E, $ in G**

**F → E+ T/T**

**T → T* F / F**

**F → id**



For More Study Materials : www.keralanotes.com

**Sol:** The closure in

$E^1 \rightarrow E, \$$

$E \rightarrow E+ T, \$$

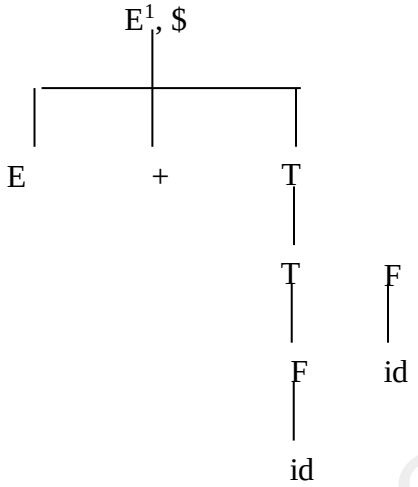
$E \rightarrow T. \$/ +$

$T \rightarrow . T* F, \$$

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

$F \rightarrow . id, \$, +, *$

The tree

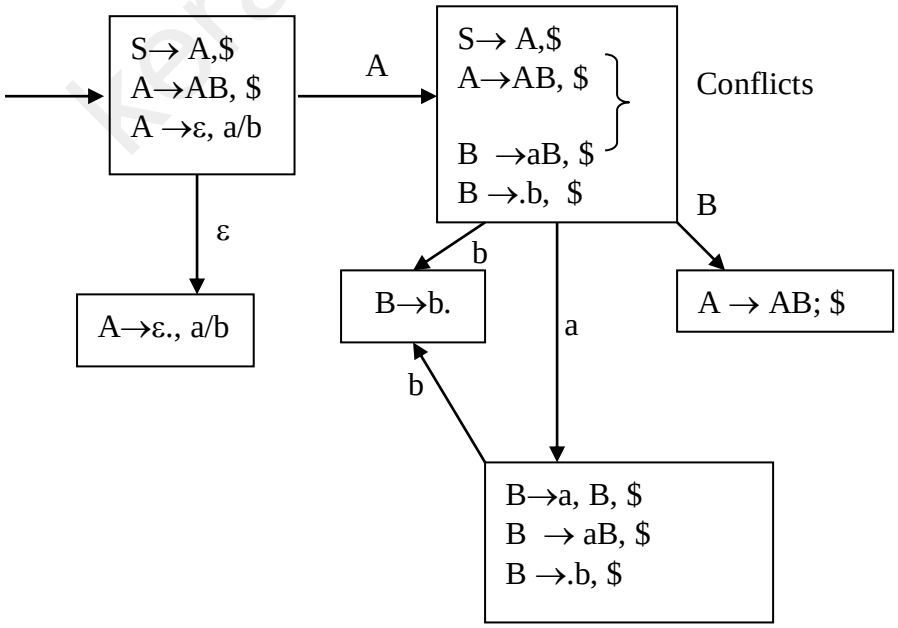



**Example 3.37**

**Find the conflicts if any in the DFA for LR (1) items**

**$S \rightarrow A,\ A \rightarrow AB/\varepsilon,\ B \rightarrow aB/b$**

**Sol:**




There is one conflict in the LR(I) machine



For More Study Materials : www.keralanotes.com

**Example 3.38**

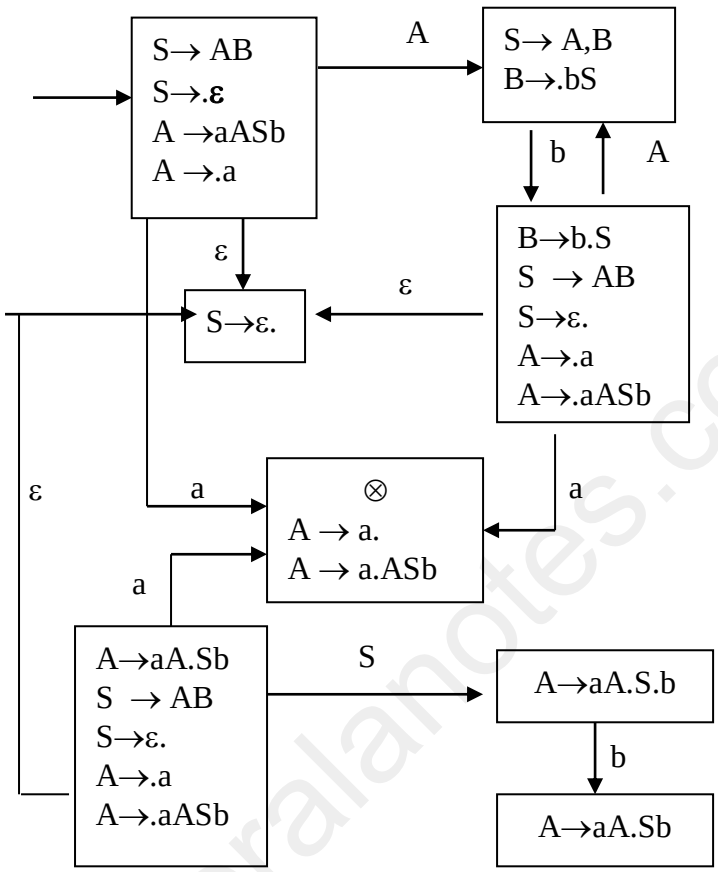
**Check if G is SLR (I)**

**S → AB/ε,**

**A→ aASb/a**

**B→ bS**

**Sol:**




The FOLLOW of A can be a or #. So we cannot resolve state⊗ . So the grammar is not SLR (I)

**Example 3.39**

**Check if G is SLR (I)**

**S → aB/ ab;**

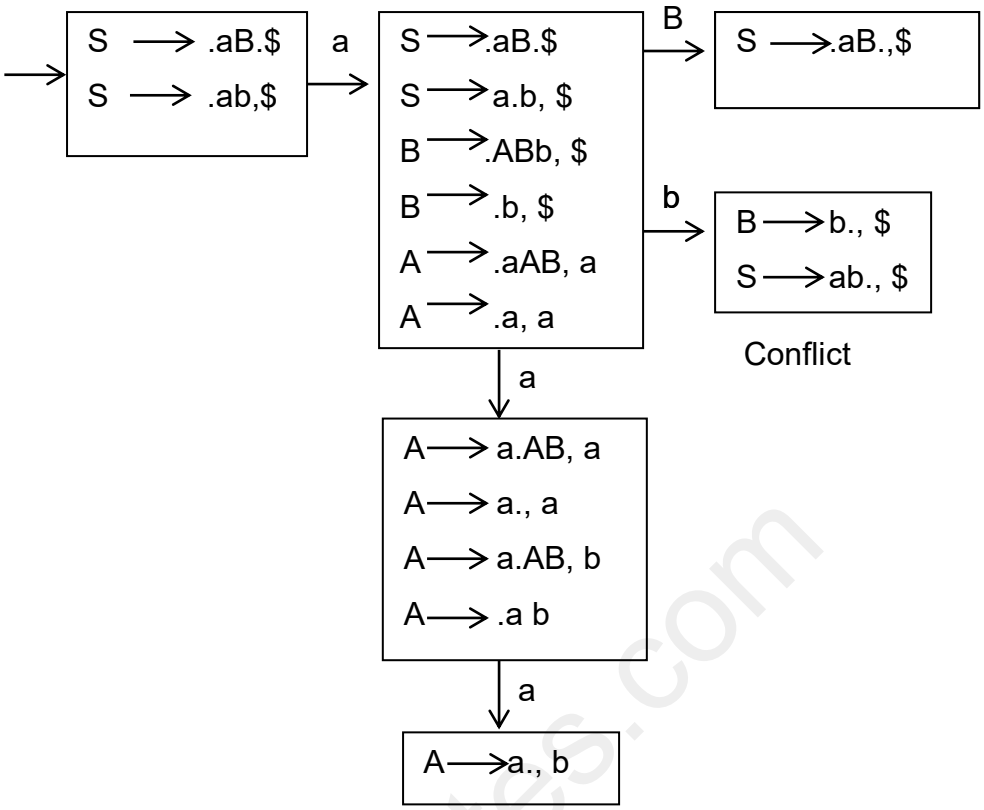
**A→ aAB/a;**

**B→ ABb/b**



For More Study Materials : www.keralanotes.com

**Sol :** Construct the LR(1) machine




A conflicting state which cannot be resolved. Not LR(1)

**Example 3.40 :**

**The LR(0) parser**

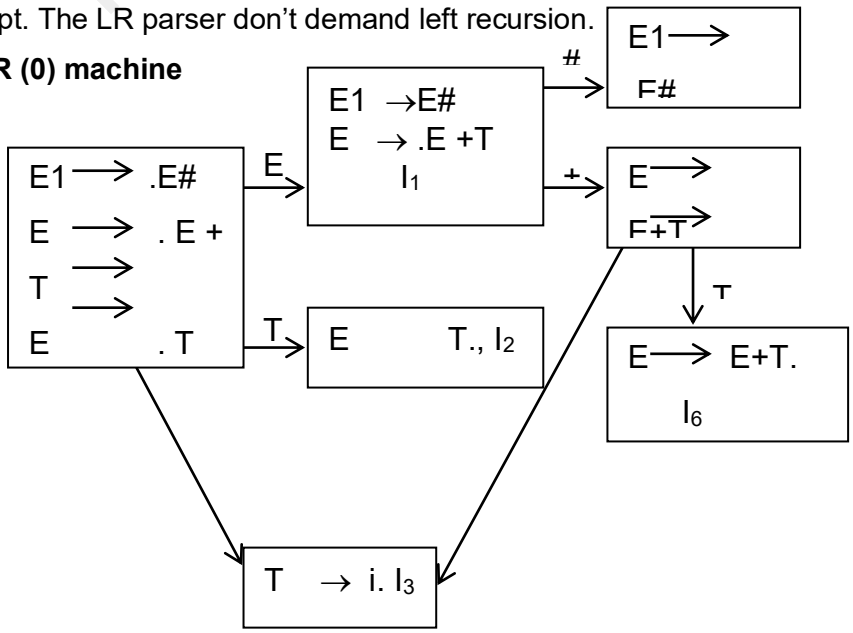
**Sample grammar**

**E1 →E#**

**E →E + T / T**

**T → i**

- Concept. The LR techniques demand unambiguous grammar
- Concept. The LR parser don't demand left recursion.
- **The LR (0) machine**






For More Study Materials : www.keralanotes.com

☞ The LR (0) parsing table

| ↓ | i | + | # | E1 | E | T |
|---|---|---|---|---|---|---|
| 0 | S3 | | | | 1 | 2 |
| 1 | | | S5 | S4 | | |
| 2 | Reduce unit E→T | | | | | |
| 3 | Reduce unit T→ i | | | | | |
| 4 | Reduce unit E1→E# | | | | | |
| 5 | S3 | S6 | | | | |
| 6 | Reduce unit E→E + T | | | | | |

As there are no shift – reduce or reduce – reduce conflicts the above grammar is LR(0)

☞ The predictive parsing table for the grammar

E1 → E #

E → E + T/T

T → i

As the E – rules are left recursive the grammar isnot LL (0)

☞ A grammar that is not LR (0)

Consider the grammar

r1. E → E#

r2. E→E + T / T (r6)

r3. T→T * F / F (r4)

r4. F → (E)

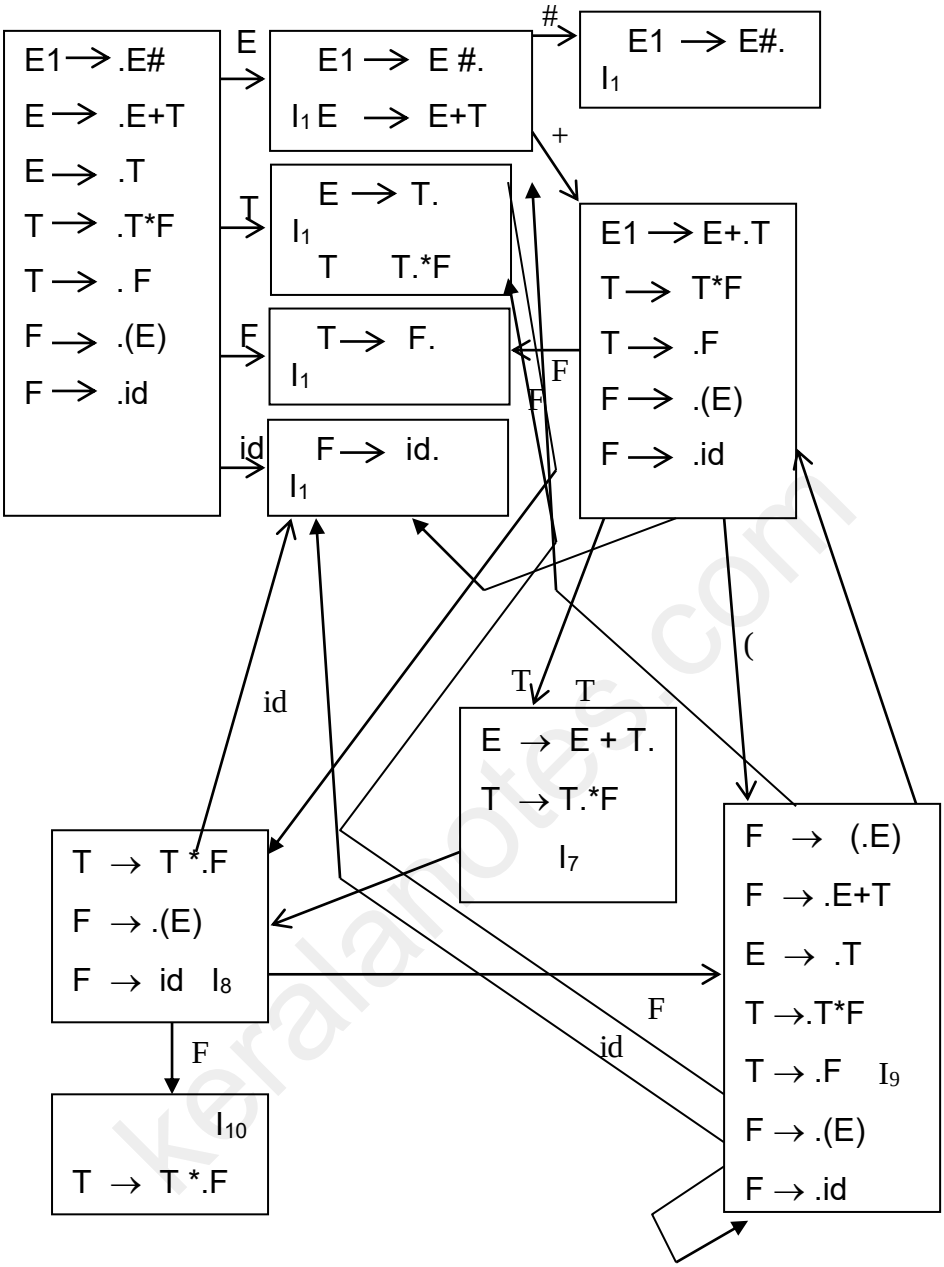
r5. F id

keralanotes.com



For More Study Materials : www.keralanotes.com

☞ The LR (0) machine for the above grammar




States 2 & 7 have a s/r conflict. So the grammar is not LR (0)

☞ The FOLLOW symbols

r1. E1 ⟶ E#

r2. E ⟶ E + T

r6. E ⟶ T

r3. T ⟶ T*F

r7. T ⟶ F



For More Study Materials : www.keralanotes.com

r4. F ⟶ (E)

r5. F ⟶ id

FOLLOW (E1) = ε

FOLLOW (E) = {#, +,)}

FOLLOW (F) = FOLLOW(T)∪FOLLOW(E) = {#, +),*}

FOLLOW(T) = {*}∪FOLLOW (T) = { #, +, *,}

☞ The SLR (1) parsing table follows

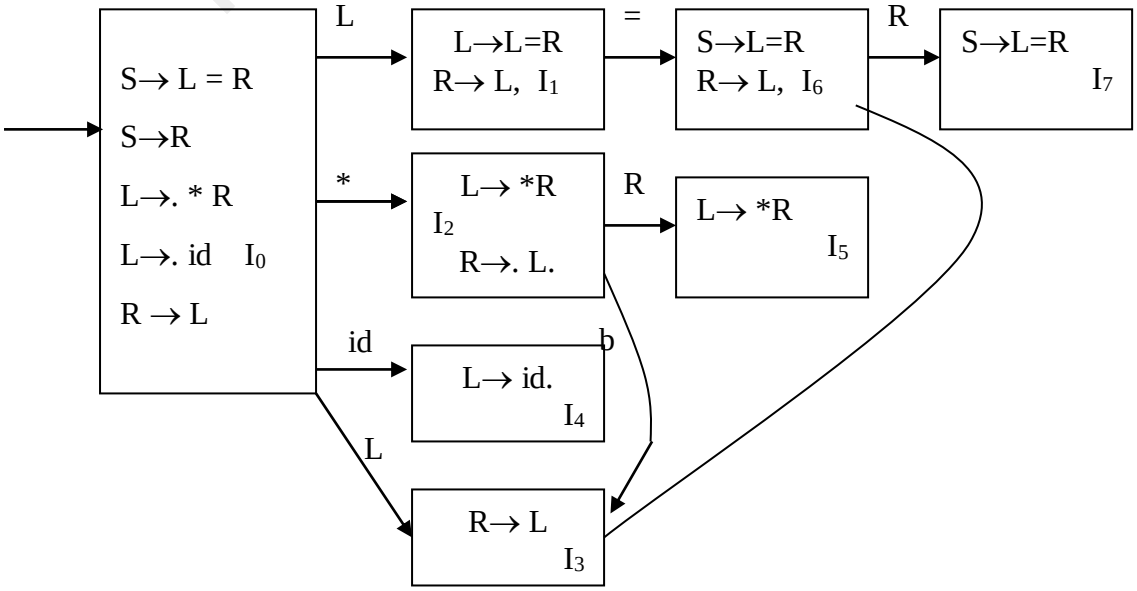
| | Id | + | * | ( | ) | # | E1 | E | T | F |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | S4 | | | | | | | 1 | 2 | 3 |
| 1 | | S6 | | | | S5 | | | | |
| 2 | | r6 | S8 | | r6 | R6 | | | | |
| 3 | | | | | | | | | | |
| 4 | | r1 | r9 | | r7 | r7 | | | | |
| 5 | Accept | | | | | | | | | |
| 6 | S4 | | | S9 | | | | | 7 | 3 |
| 7 | | r2 | | S8 | | | | | | |
| 8 | | | | S9 | | | | | | 10 |
| 9 | | | | S9 | | | | | 2 | |
| 10 | | r3 | r3 | | r3 | r3 | | | | |
| 11 | | | | | | | | | | |

**Example 3.41 :**

**A grammar that is not SLR (1)**

**S→L = R, S→R. L → *R, L →id, R →L**

☞ The LR (0) machine for the grammar






For More Study Materials : www.keralanotes.com

**Example 3.42 :**

**The LR (1) parsing technique**

**Example grammar**

r1. $S^1 \rightarrow S$

r2. $S \rightarrow C\ C$

r3. $C \rightarrow c\ C$

r4. $C \rightarrow d$

☞ The language generated by the grammar

$S \rightarrow S$

$S \rightarrow C\ C$

$C \rightarrow c\ C/d \Rightarrow C^*\ d\ C^*\ d$

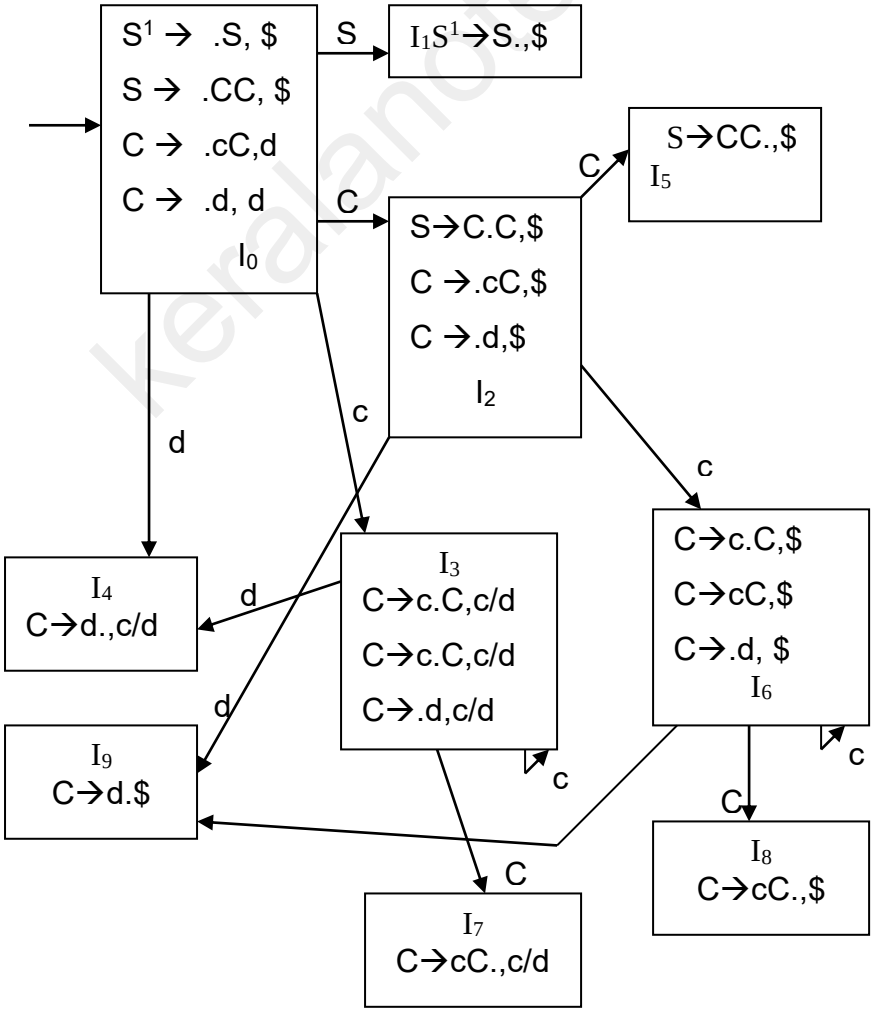
So the language generated by the grammar is a regular set.

☞ The LR(1) machine for the grammar

r1 $S^1 \rightarrow S$

r2 $S \rightarrow C\ C$

r3 & r4 $C \rightarrow c\ C/d$






For More Study Materials : www.keralanotes.com

☞ The LR(1) parsing table

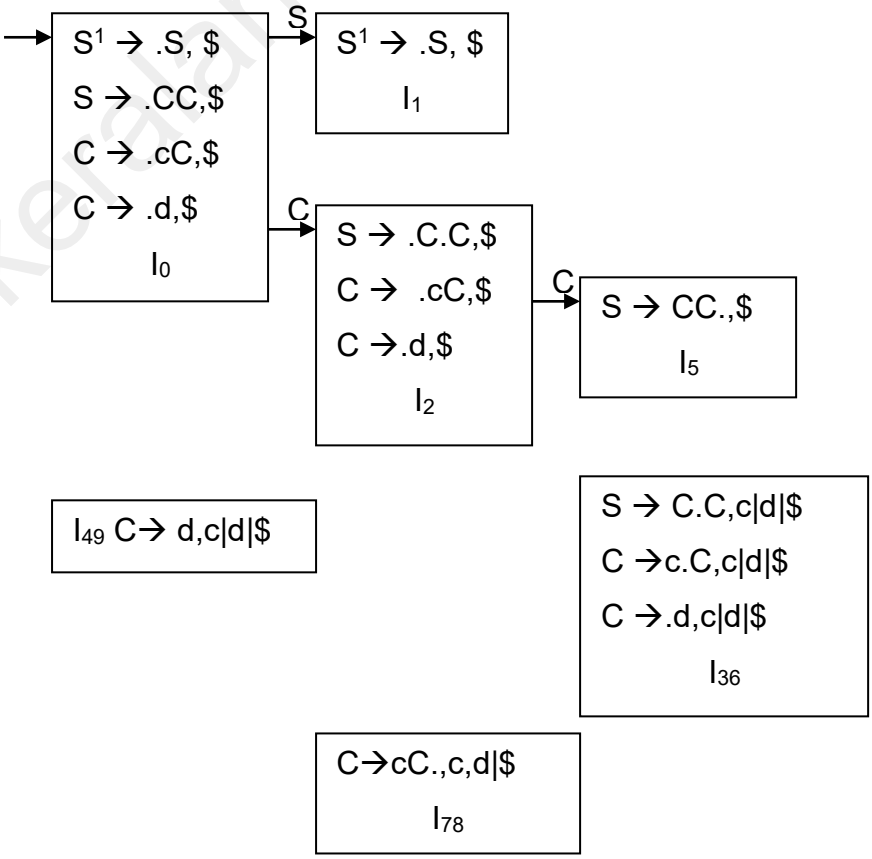
| | c | d | $ | $^1$ | S | C |
|---|---|---|---|---|---|---|
| 0 | S3 | S4 | | | 1 | 2 |
| 1 | | | r1 | | | |
| 2 | | | | | | 5 |
| 3 | S3 | S4 | | | | 7 |
| 4 | r4 | r4 | | | | |
| 5 | | | r2 | | | |
| 6 | S6 | S9 | | | | 8 |
| 7 | r3 | r3 | | | | |
| 8 | | | r3 | | | |
| 9 | | | | r4 | | |

☞ The LALR(1) machine

r1 $S^1 \rightarrow S$

r2 $S \rightarrow C\ C$

r3 & r4 $C \rightarrow c\ C/d$




keralanotes.com



For More Study Materials : www.keralanotes.com

☞ The LALR(1) Parsing table

| | c | D | $ | $S^1$ | S | C |
|---|---|---|---|---|---|---|
| 0 | | | S49 | | 1 | 2 |
| 1 | | | r1 | | | |
| 2 | S36 | | S49 | | | 5 |
| 36 | S36 | S49 | | | | 78 |
| 49 | r4 | r4 | r4 | | | |
| 5 | | | r3 | | | |
| 78 | r3 | r3 | r3 | | | |

**Example 3-43 :**

**Consider**

**$S^1 \rightarrow S$**

**S → a Ad/b Bd/ a Bc/ b Ac**

**A→c**

**B →c**

☞ Is the grammar LL(1)?

FIRST (S) = {a, b}

If S is the goal & a or b are the incoming symbols, then there is a conflict of LL(1).

So no predictive parsimg table exist without conflicts.

☞ The predictive parsing table

| | a | b | c | d | $ |
|---|---|---|---|---|---|
| $S^1$ | $S^1 \rightarrow S$ | $S^1 \rightarrow S$ | | | |
| S | S →aAd | S→bBd | | | |
| | S →aBc | S→bAc | | | |
| A | | | | A →c | |
| B | | | | B→c | |

keralanotes.com



For More Study Materials : www.keralanotes.com

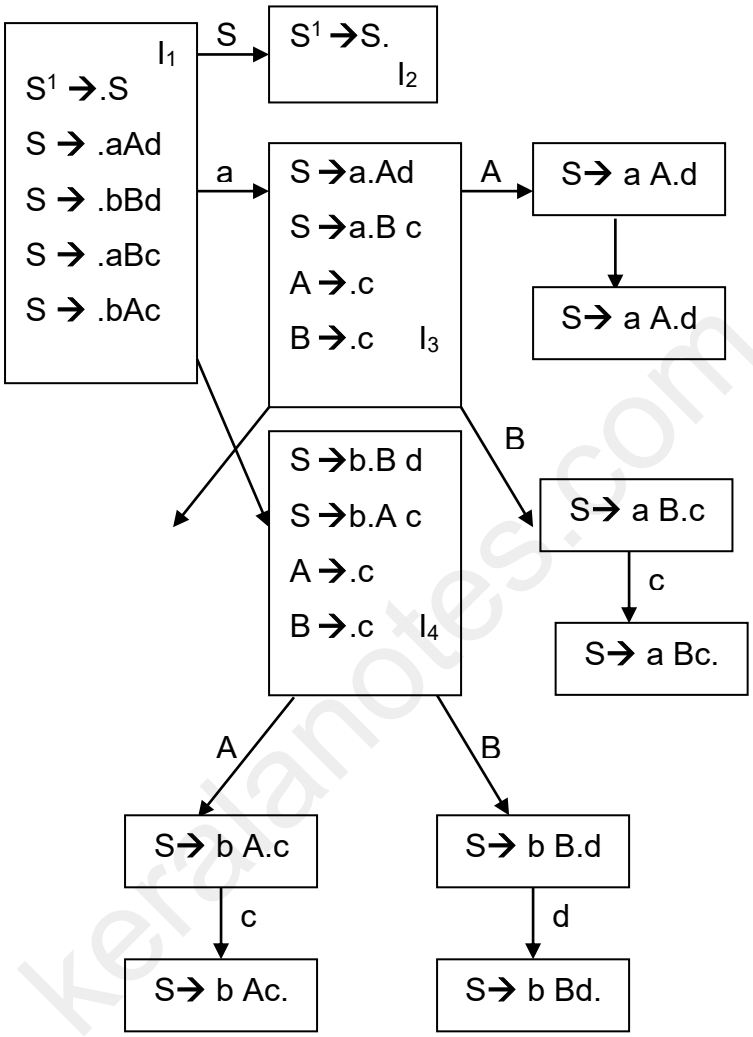
☞ The LR(0) parsing table for a sample grammar

$S^1 \rightarrow S$

$S \rightarrow aAd/\ bBd/\ aBc/\ bAc$

$A \rightarrow c$

$B \rightarrow c$




The grammar is not LR(0)

☞ The SLR (1) parsing table for a sample grammar

$S^1 \rightarrow S$

$S \rightarrow aAd/\ bBd/\ aBc/\ bAc$

$A \rightarrow c$

$B \rightarrow c$

The FOLLOW sets

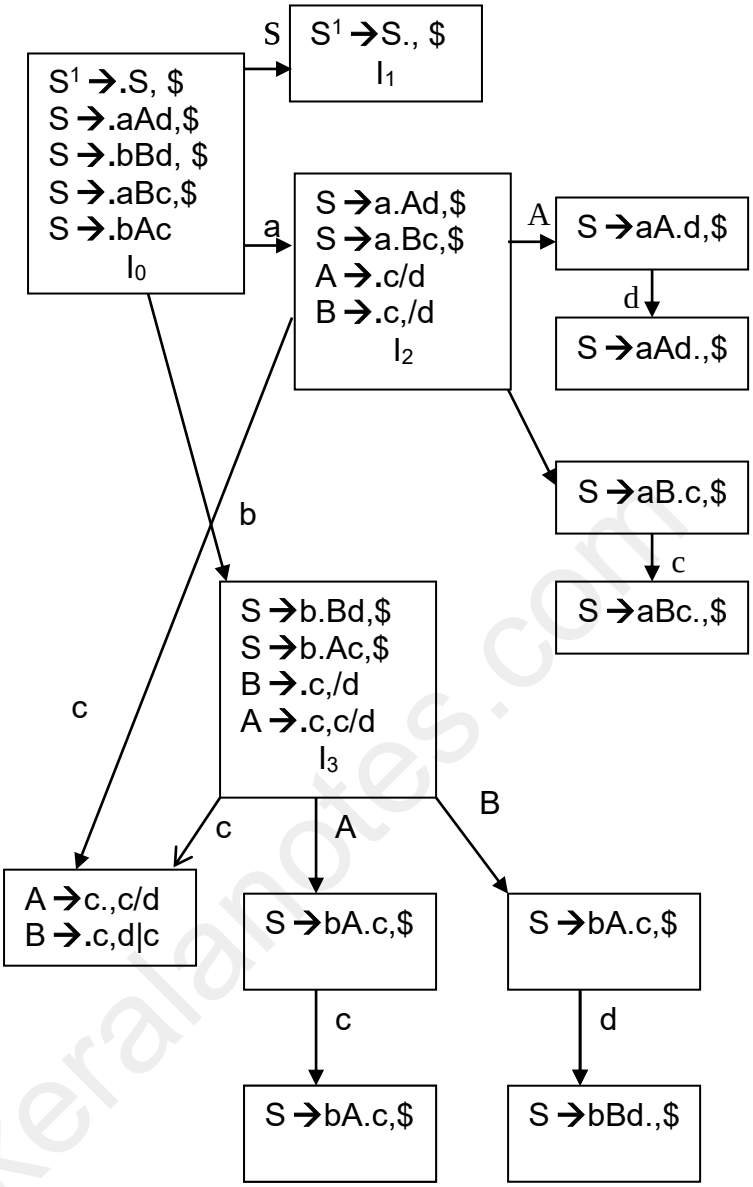
FOLLOW($S^1$) = {$}, FOLLOW (S) = {$}

FOLLOW(A) = {d, c}



For More Study Materials : www.keralanotes.com

The SLR (1) machine




The grammar is not SLR (1)

☞ The LR (1) parsing technique for a sampl grammar

r1 : $S^1 \rightarrow S$

r2, r3, r4, r5 : S $\rightarrow$ aAd/ bBd/ aBc/ bAc

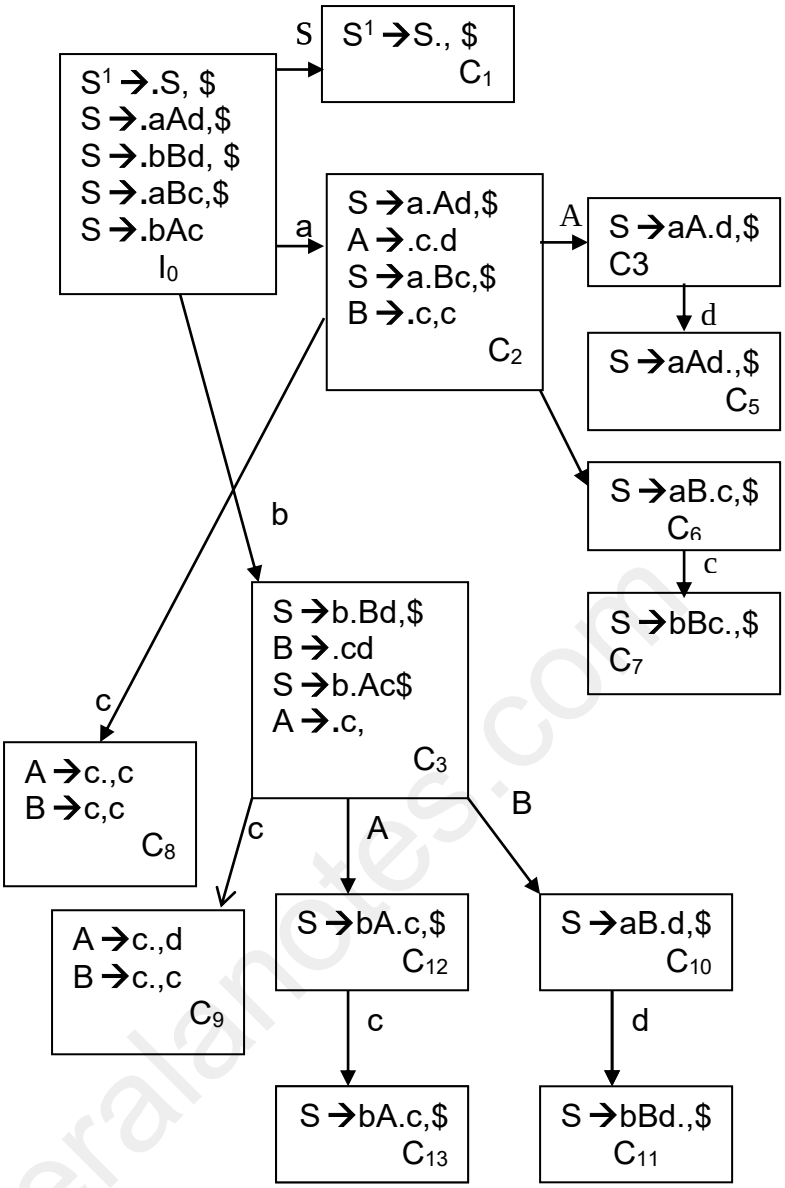
r6 : A $\rightarrow$ c

r7 : B $\rightarrow$ c

The LR (1) machine



For More Study Materials : www.keralanotes.com




keralanotes.com



For More Study Materials : www.keralanotes.com

The grammar is LR(1) but not LALR (1)

☞ The LR (1) parsing table

| | a | b | c | d | $ | S' | A | B | S |
|---|---|---|---|---|---|---|---|---|---|
| 0 | S2 | S4 | | | | | | | 1 |
| 1 | | | | | r1 | | | | |
| 2 | | | S8 | | | | 3 | 6 | |
| 3 | | | | S5 | | | | | |
| 4 | | | | | | | | | |
| 5 | | | S9 | | | | 12 | 10 | |
| 6 | | | S7 | | | | | | |
| 7 | | | | | r4 | | | | |
| 8 | | r7 | r6 | | | | | | |
| 9 | | r6 | r7 | | | | | | |
| 10 | | | | S11 | | | | | |
| 11 | | | | | r2 | | | | |
| 12 | | | S13 | | | | | | |
| 13 | | | | | r4 | | | | |

The grammar is LR (1).

As states 8 & 9 cannot be merged

The grammar is not LALR (1).

**Example 3 -44 :**

**Consider the grammar**

**S → Aa |bAc| dc | bda**

**A → d**

**(a) The grammar is LL (1)** **(b) The grammar is LR (0)**

**(c) The grammar is SLR (1)** **(d) The grammar is LALR (1)**

**Ans : (d)**

**Sol :** The rules S →Ac/ bda show that a conflict arises when S is the goal and b is the incoming symbol. The grammar is thus not LL (1)

We will construct the LR (0) machine & show that the conflicts cannot be resolved based on the FOLLOW set. So the grammar is not SLR (1). We will construct a LR (1) machine for the grammar. We will merge sets with common cores & show that the resulting grammar LALR (1) has no conflicts.

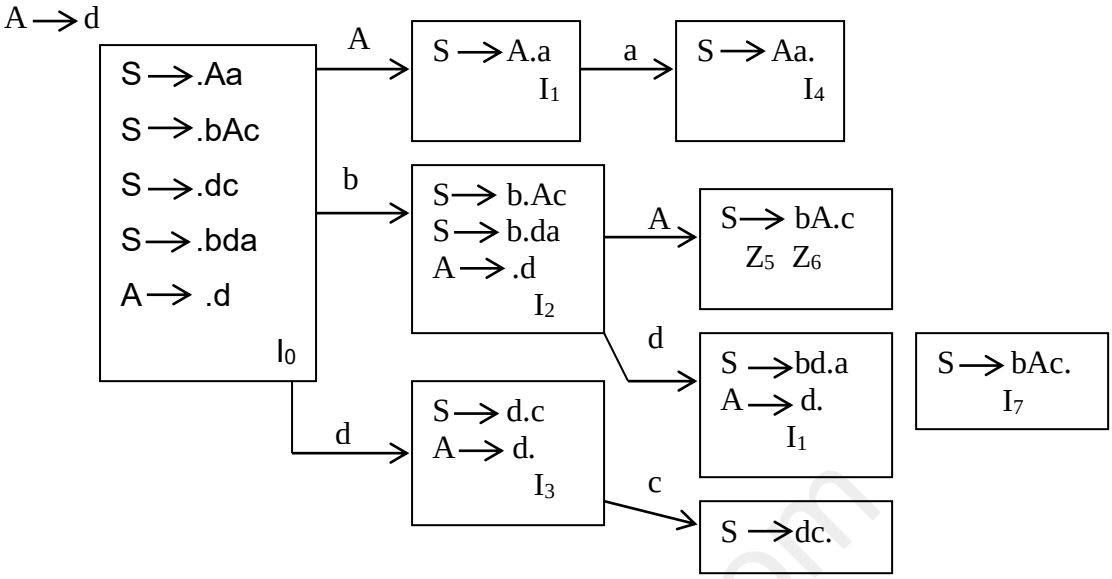


For More Study Materials : www.keralanotes.com

The LR (0) machine for the grammar

S → Aa/bAc/dc/bda

A → d




State $I_3$ has a s/r conflict, and state $I_7$ has a r/r conflict. So the grammar is not LR(0).

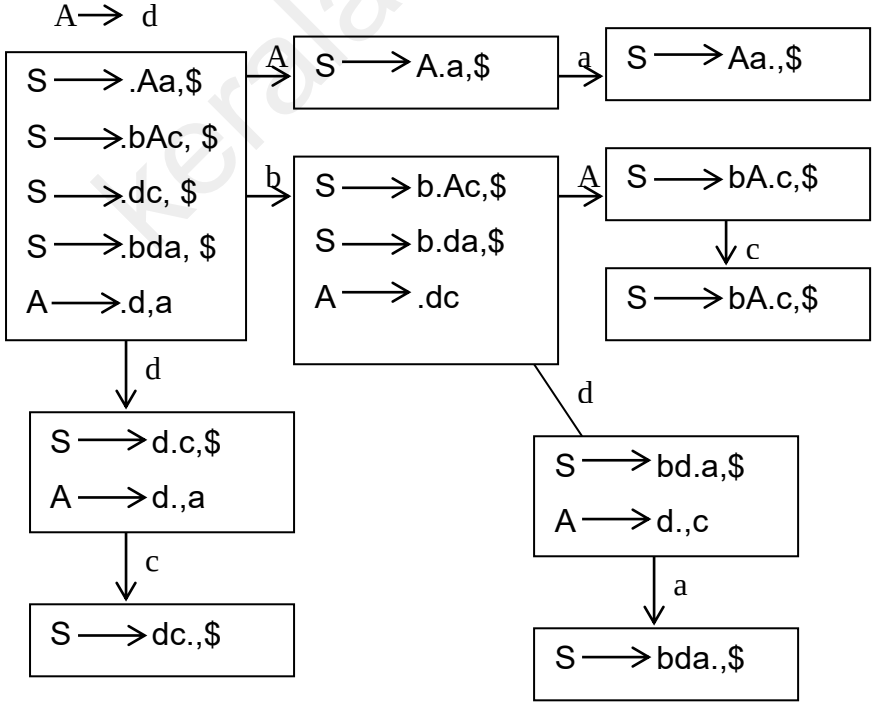
Can the grammar be SLR (1)?

FOLLOW(S) = {$} and this does not resolve the conflicts $I_7$. So the grammar is not SLR(1)

The LR(1) machine for the grammar

S → Aa/bAc/dc/bda

A → d






For More Study Materials : www.keralanotes.com

The grammar is LR (1) and as there are no two states with a common core, the grammar is alos LALR(1).

**Example 3.45 :**

**Consider the grammar**

**S ⟶ Aa/bAc/Bc/bBa**

**A ⟶ d**

**B ⟶ d**

**(a) The grammar is LL (1)** **(b) The grammar is LR (0)**

**(c) The grammar is SLR (1)** **(d) The grammar is LR (1) but not LALR(1)**

**Ans : (d)**

**Sol :** The rules S ⟶ bAc| bBa show that when S is the goal & b is the incoming symbol we cannot choose the correct rule. So the grammar is not LL (1).

We will construct the LR(0) machine & show that it has conflicts. So the gramamr cannot be LR(0). The inadequate states cannot be restricted by the FOLLOW sets, so the grammar is not SLR(1).

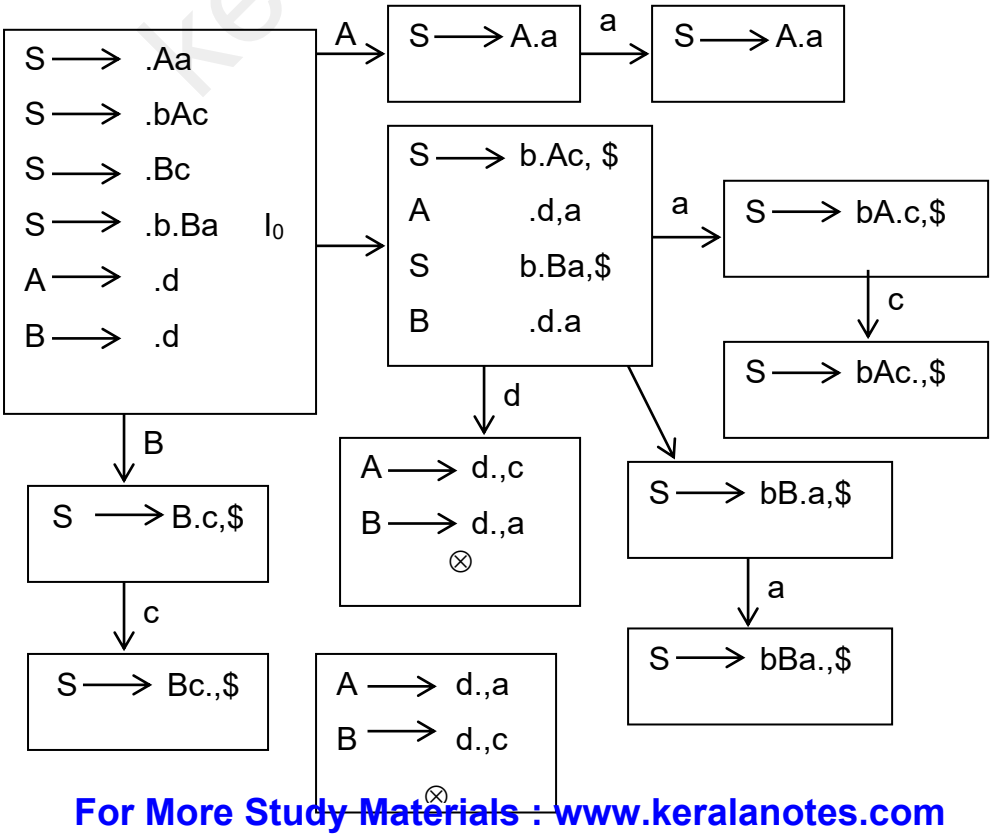
The grammar will be shown to be LR(1) by constructing a LR(1) machine for the same. Maerging state with common cores leads to conflicts to the grammar will not be LALR(1).

The LR(0) machine.

S ⟶ Aa|bAc| Bc| bBa

A ⟶ d

B ⟶ d






For More Study Materials : www.keralanotes.com

As the LR (1) machine has no conflicts, the grammar is LR(1). If state is ⊗ & ⊗ which have a common core are merged, conflicts arise. So the given grammar is not LALR (1).

**Example 3.46 :**

**Consider the following grammar**

**E ⟶ T { + T}**

**T ⟶ a**

**Which one of the following is the proper recursive descent parser for the grammar?**

(a)
```
void E ( )
{
    T ( );
    While (input = '+')
    Scan ( ); T( );
}
Void T ( )
{
    Is (input? = a) error ( );
    Scan ( ); T ( );
{
    Is (input? = a) error ( );
    Scan ( );
}
```

(b)
```
void E ( )
{
    T ( ) ; T ( ) ;
    Scan ( );
}
Void T ( )
{
    Scan ( )
}
```

(c)
```
void E ( )
{
    If (input = '+') scan;
    T ( ); T ( );
}
Void T ( );
{
    Scan ( );
}
```

d) None of the above

**Ans : (a)**

**Sol :** Trace the syntax graph.

keralanotes.com



For More Study Materials : www.keralanotes.com

**Example 3.47 :**

**The Chomsky Normal Form grammar for**

**S → (L)/ a;  L → L, S/S**

(a) S → AB/a
B → LC
A → (
C → )
L → AB
L → LD
D → ES
E → ,

(b) S → LL/ a
L → LS/,

(c) S → AB/a
B → LD
A → (
C → )
L → AB
L → LD
E →.

(d) None of the above

**Ans : (a)**

**Sol :** S → AB/a
A → (
B → L )
L → LD
D →, S
L → AB

S → AB/C
A → (
C → )
B → LC
L → AB
L → LD
D → ES
E →,

**Example 3.48 :**

**Consider the grammar**

**S A S/b**

**A S A/a**

(a) The grammar is ambiguous
(b) The grammar is LL (1)
(c) The grammar is SLR (1) but not LR (0)
(d) The grammar is LR (1) but not LALR(1)

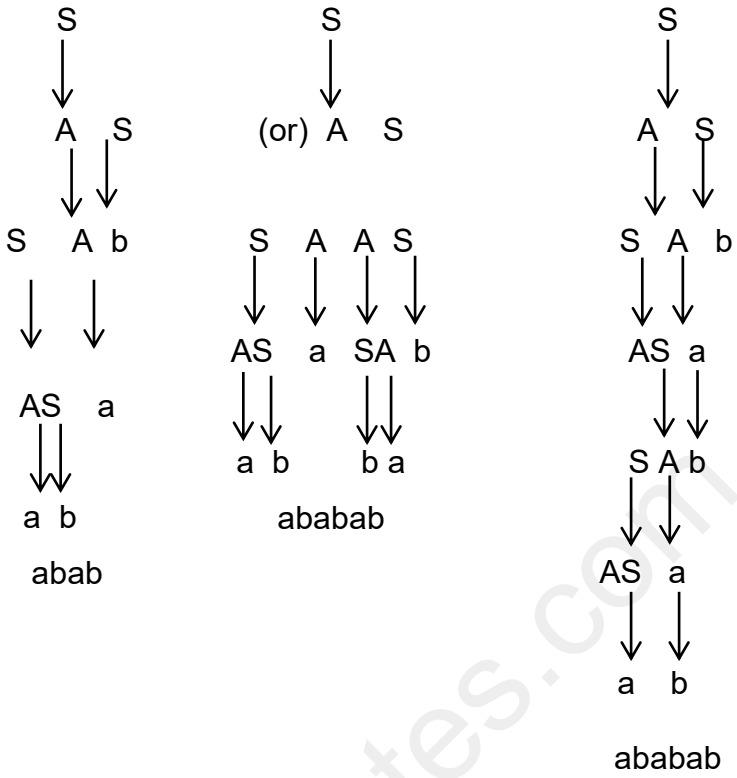
keralanotes.com



For More Study Materials : www.keralanotes.com

**Ans : (a)**

**Sol :** Consider the string abab



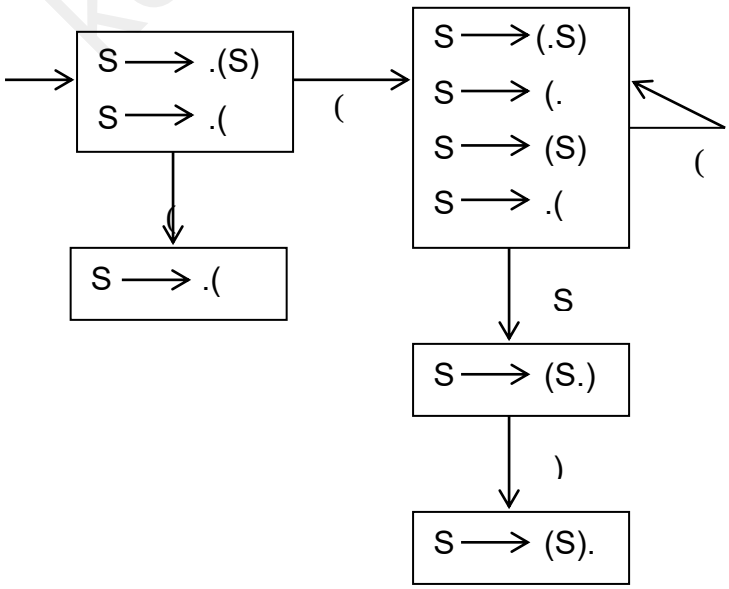

**Example 3.49 :**

**Compute the LR (0) item sates & then transitions for the grammar**

**S ⟶ (S)**

**S ⟶ (S)**

**Sol :**




keralanotes.com



For More Study Materials : www.keralanotes.com

The LR(0) item sets are

1) S ⟶ .(S)
S ⟶ . (

2) S ⟶ (.

3) S ⟶ (.S)
S ⟶ (.
S ⟶ . (S)
S ⟶ . (
S ⟶ x S x / S

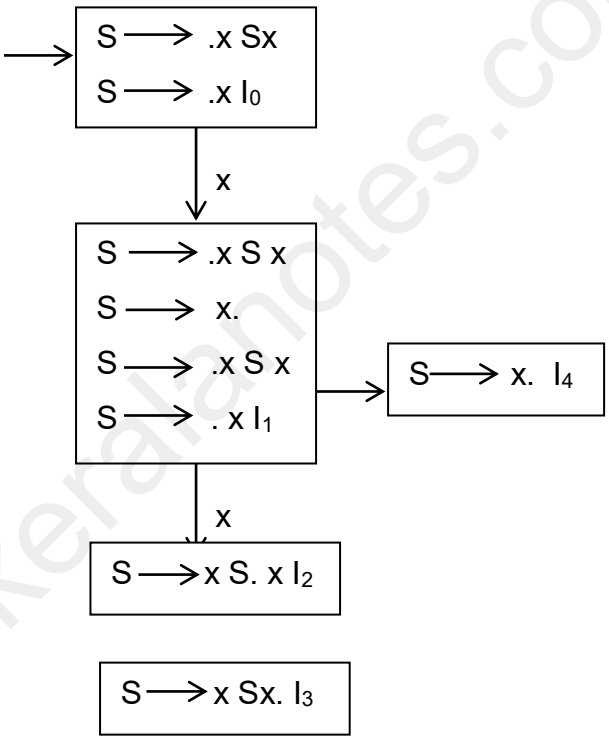
4) S ⟶ (S.)

5) S ⟶ (S).

**Example 3.50 :**

**Construct the LR (0), SLR(1), LR(1) & LALR(1) automats for the grammar.**

**Sol :** The LR (0) machine




There is a S/R conflict in $I_1$. This can not be resolved by the FOLLOW symbol x. So the grammar is not SLR (1).

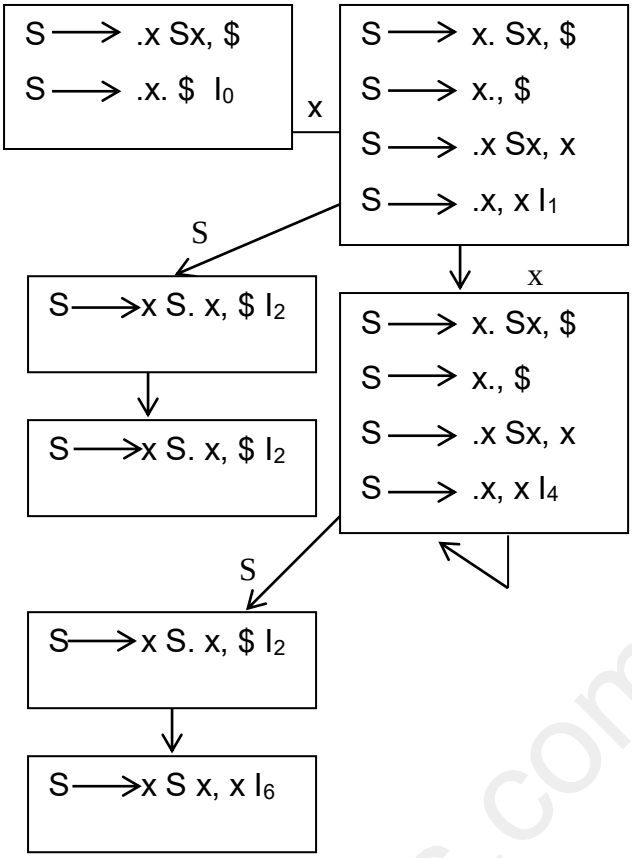
The LR (1) machine for the grammar

S ⟶ x S x

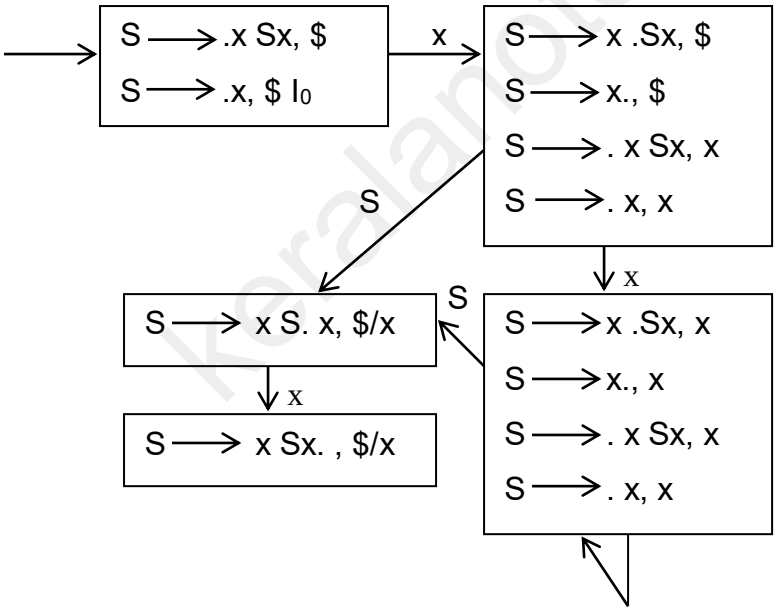
S ⟶ x



For More Study Materials : www.keralanotes.com




The LALR(1) machine




keralanotes.com

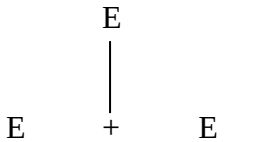


For More Study Materials : www.keralanotes.com

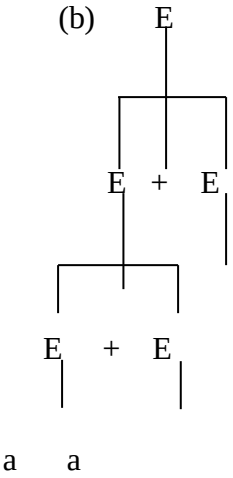
**Classroom Practice Questions**

01. Let + be a left associative operator then the syntax tree for a + a + a is in the grammar E E + E/a ——

(a)




(b)




(c) Either (a) or (b) (d) Neither (a) nor (b)

02. Consider the grammar given below

S → SS|a|ε

The sentence "aaa'' has

(a) a unique derivation tree in the grammar

(b) two leftmost derivations in the grammar

(c) ten rightmost derivations in the grammar

(d) and infinite number of leftmost and rightmost derivations

03. A grammar that is both left and right recursive for a non-terminal is

(a) Ambiguous (b) Unambiguous

(c) Information is not sufficient (d) None

04. A grammar has the following productions :

S → a Sb/b/bSa

Which of the following sentences are in the language that is generated by this grammar

(a) aaaaabb (b) aabbaabb

(c) bbbaabbaa (d) ababbab

05. Consider the following grammar

E ⟶ E + E / (E * E)/id

The number of parse trees are possible for an input string w = id+id+id is_____.



For More Study Materials : www.keralanotes.com

06. How many parse trees exists for an input string w = abab in the CFG

S ⟶ aSbS/bSaS/ε

(a) 4 (b) 3 (c) 2 (d) 5

07. Consider the left recursive grammar

A ⟶ Sa/b

S ⟶ Sc/Ad

Which of the following grammar is equivalent grammar with left recursion eliminated?

(a) A ⟶ Sa/b
S ⟶ bbS'
S' ⟶ cS'/adS'/ε

(b) A ⟶ Sa/b
S ⟶ bdS'
S' ⟶ cS'/adS'

(c) S ⟶ bdS'
S' ⟶ cS'/adS'/ε
A ⟶ ScaA'/bA'
A' ⟶ daA'/εc

(d) none

08. Consider the following expression grammar G :

E ⟶ E – T | T
T ⟶ T + F | F
F ⟶ (E) | id

Which of the following grammars is not left recursive, but is equivalent to G?

(a) E ⟶ E – T | T
T ⟶ T + F | F
F ⟶ (E) | id

(b) E ⟶ TE'
E' ⟶ -TE'|ε
T T + F | F
F ⟶ (E) | id

(c) E ⟶ TX
X ⟶ - TX| ε
T ⟶ FY
F ⟶ (E) | id

(d) E ⟶ TX | (TX)
X ⟶ - TX | + TX | ε
T ⟶ id

09. Left factoring the grammar helps in

(a) Removing the Ambiguity (b) Avoiding going into infinite loop

(c) Avoiding Back Tracking (d) None

keralanotes.com



For More Study Materials : www.keralanotes.com

10. Consider the following grammar

E → E ↑T / T

T → F + T / F

F → id

(a) ↑ has higher precedence than +

(b) + has lower precedence that ↑

(c) ↑ has lower precedence than +

(d) id has lower precedence than ↑ and +

11. Given the following grammar

E → E*F/F+E/F

F → F-F/id

Which of the following is **true?**

(a) * > +  (b) - > *

(c) + = -  (d) + > *

12. Consider the following left – associative operators, in decreasing order of precedence ___, $ *(exponent), *. The result of the expression.

3 – 2* 4 $ 2* 3 $ 2 is _____.

13. Evaluate the following expression considering the given rules :

3 – 2* 4 $ 1 * 2 $

Where $ mean exponentiation

(a) (-) highest followed by (*) and then followed by($) and all are left associative

(b) (-) highest followed by (*) and then followed by ($) and all are right associative

(c) ($) highest followed by (-) and then followed by (*) and all are left associative

(a) a<b<c  (b) c<a<b  (c) a<c<b  (d) b<c<a

14. Choose the correct statement?

(a) a bottom up parsing technique simulates a rightmost derivation

(b) a top down parsing technique simulates the reverse of a leftmost Derivation

(c) a bottom up parsing technique building the derivation tree in bottom up and simulates a rightmost derivation in reverse.

(d) a top down parsing technique simulates building the derivations tree in top down and simulates a leftmost derivation in reverse.



For More Study Materials : www.keralanotes.com

15. What is FIRST (S) in

S ⟶ AC/ Ca/ Bb,

A ⟶ d

B ⟶ e/ε

C ⟶ f/ε

(a) {a, d, e, f} (b) {a, e, f, ε}

(c) {a, b, d, e, f} (d) {a, b, d, e, f, ε}

16. Consider the following grammar for expressions :

<expression> ⟶ atom <additional> | (<expression>)

<additional> ⟶ +expressional>|*<expression>|ε

The follow set for < additional > is

(a) { atom, ( } (b) { [, $}

(c) {+, *, $} (d) {$}

17. S ⟶ ABC

A ⟶ aA|c

B ⟶ b|ε

C ⟶ c

FIRST (A) ∩ FOLLOW (A) is

(a) {a} (b) {b}

(c) {c} (d) none

18. FOLLOW (B) in the following grammar

A ⟶ BCD

B ⟶ w|Bx

C ⟶ yCz/m

D ⟶ DB|a

(a) {x, y, m, $} (b) {x, y, w, $}

(c) {x, w, m, $} (d) {x, y, m, w, $)

19. S ⟶ [SX] |a

X ⟶ ε|+SY|Yb

Y ⟶ ε|=SXc

Then FOLLOW (S) is

(a) {$, +, -, ], c, b} (b) {$, _, ], c, b}

(c) {$, +, _, ], b } (d) {$, ], c, b}

keralanotes.com



For More Study Materials : www.keralanotes.com

20. In the grammar S ⟶ TA,

A ⟶ +TA/ε

T ⟶ a/ε

Follow (T) ∩ First (S) is

(a) {+, ε} (b) {+, $}

(c) {+} (d) {a, +, ε}

21. Consider the following grammar

S ⟶ AaC/Bd

A ⟶ BC

B ⟶ bB/C

C ⟶ accS

For which non terminals is follow set contains terminal 'a'.

(a) {C} (b) {A} (c) {A, B, C} (d) {A, B, C, S}

22. Consider the following grammar

<expression> ⟶ <factor> <rest>

< rest > ⟶ * < expression> | ε

< factor > ⟶ identifier

In the predictive parsing table M of the grammar the entries for

M [<expression>, identifier] and M [<rest>, *] respectively are

(a) [<expression> ⟶ < factor> <rest>] and [<rest> ⟶ ε]

(b) [<expression> ⟶ factor> <rest>] and []

(c) [<expression> ⟶ factor> <rest>] and [< rest> ⟶ * <expression>]

(d) [<factor> ⟶ identifier] and [<rest> ⟶ ε]

23. Consider the grammar

S ⟶ a | ab | abc

Choose the correct statement from the following :

(a) The grammar is LL (1) (b) The grammar is LL (2)

(c) The grammar is LL (3) (d) None of the above

24. Consider the grammar

S ⟶ if expr then

| if expr then stmt

| if expr then stmt else

| if expr then stmt else stmt



For More Study Materials : www.keralanotes.com

The grammar is

(a) LL (1) (b) LL (4)

(c) LL (5) (d) LL (6)

25. Consider the grammar, G1

E ⟶ E + E|E*E|(E)|id

Consider the grammar G2,

E ⟶ E + T|T

T ⟶ T*F|F

F ⟶ (E)| id

Choose the incorrect statement?

(a) G2 generates the same language as G1

(b) G2 is unambiguous but not LL (1) as it is left recursive

(c) G2 gives a higher precedence to + over *

(d) None

26. Choose the true statement?

(a) If a grammar is left recursive it cannot be LL(1)

(b) If a grammar is right recursive it cannot be LL (1)

(c) An ambiguous grammar can sometimes be LL (1)

(d) If a grammar is not context free then it sometimes can be LL (1)

27. S ⟶ Aa

A ⟶ b/ε

If we construct a predictive parse table for the above grammar, the production A ε is added in 'A' row and _____.

(a) '$' column (b) 'ε' column

(c) 'a' column (d) 'b' column

28. S ⟶ aSbS/bSaS/ε

In the predictive parse table M of the above grammar M [S,a] = _____

(a) S ⟶ aSbS (b) S ⟶ bSaS

(c) S ⟶ ε (d) S ⟶ aSbS, S ⟶ ε

29. S ⟶ Aa/ Bb

A ⟶ b

B ⟶ b is

(a) LL(1) (b) LL(2) (c) LR (0) (d) None



For More Study Materials : www.keralanotes.com

30. Consider a calculator modeled by the grammar

<accumulated _ sum > ⟶ <accumulated_sum>+number
|<accumulated_sum>*number
|number

For an input "number + number*number", the handles in the reverse of a rightmost derivation are

(a) number, <accumulated_sum>+number.
<accumulated_sum>+number*number

(b) number, <accumulated_sum>+number
<accumulated_sum>+<accumulated_sum>*number

(c) number, number +number, number + number* number

(d) number, <accumulated_sum>+number, <accumulated_sum>* number

31. Which of the following is not an operator grammar?

I. S ⟶ AB  II. S ⟶ AaB

III. S ⟶ a  IV. S ⟶ ε

(a) only I  (b) I and II

(c) I, III and IV  (d) None

32. Which of the following is not an operator grammar

(a) S ⟶ AaB
A ⟶ aA/b
B ⟶ bB/a

(b) S ⟶ AaB
A ⟶ a/b

(c) S ⟶ AaB
A ⟶ aA/b
B ⟶ bB/ε

(d) None

33. What is equivalent operator grammar for the following grammar?

S ⟶ AB, A ⟶ c/d, B ⟶ aAB/d

(a) S ⟶ Aa/Ab
A ⟶ c/d

(b) S ⟶ AaAB/Ad
A ⟶ c/d
B ⟶ aAB/d

(c) S ⟶ AaS/Ab
A ⟶ c/d
B ⟶ aS/b

(d) None



For More Study Materials : www.keralanotes.com

34. In operator precedence parsing, precedence relations are defined

(a) for all pairs of non – terminals

(b) for all pairs of terminals

(c) to delimit the handles

(d) both (b) and (c)

35. Consider the grammar given below

E → E+E|E*E|E-E|E/E|E^E|(E)|id

Assume that + and – have the same but least precedence, * and / have the next higher precedence but the same precedence and finally^has the highest precedence. Assume + and – associate to the left like * and / and that ^ associates to the right. Choose the correct statement in the operator precedence table constructed for the grammar the relations for the ordered pairs $(^{\wedge},^{\wedge}),(^{-},^{-}),(^{+},^{+}),(^{*},^{*})$ are

(a) all < (b) all >

(c) <,>,=,< (d) <,>,>,>

36. S → AaBb

A → Bc/D

B → e

In the operator grammar above lead (S) = _____

(a) {d} (b) {e, d}

(c) {c, d, e} (d) {a, c, d, e}

37. Consider the following operator grammar

E → E + T/T

T → F↑T/F

F → id

(a) ↑ >↑ (b)↑> +

(c) + >↑ (d) + < +

38. Choose the correct statement?

(a) All (1) grammar is compulsory SLR (1)

(b) An operator grammar can be ambiguous but a parser may be Possible

(c) An operator grammar is recursive LR(1) always

(d) An grammar may be LALR (1) but not LR(1)



For More Study Materials : www.keralanotes.com

39. Choose the correct statement?
    (a) A grammar that is LR(0) may not be LL (1)
    (b) A grammar that is LR is necessarily LL (1 that is LR is necessarily LL (1)
    (c) A SLR(1) grammar is always LR(0)
    (d) None of the above
40. Choose the false statement?
    (a) An ambiguous grammar can never be LL (1)
    (b) An ambiguous grammar can never be LR (0)
    (c) An ambiguous grammar ca n never be SLR (1)
    (d) An ambiguous grammar cannot have a shift reduce parser
41. Choose the false statement?
    (a) There exists a grammar that is LR(0) but not LL(1)
    (b) There exists a grammar that is LL (1) but not LR (0)
    (c) There exists a grammar that is LL(1) but not SLR (1)
    (d) There exists an ambiguous grammar that is either LL(1) or LR(0)
42. Choose the false statement?
    (a) Top down parsing algorithms simulate a leftmost derivation
    (b) Bottom up parsing algorithms simulate the reverse of a rightmost Derivation
    (d) An LL(1) parser is a top down parser
    (e) An LR (1) parser may sometimes exist for some ambiguous grammars
43. Choose the correct statement?
    (a) A grammar that is LL(1) is necessarily LR(0)
    (b) A grammar that is LL(2) is necessarily LR(0)
    (c) A grammar that is LL(1) is necessarily LL(2)
    (d) None of the above
44. Choose the false statement?
    (a) There exists a grammar that is SLR(1) but not LR(0)
    (b) There exists a grammar that is LALR(1) but not SLR(1)
    (c) There exists a grammar that is LR(1) but not LALR (1)
    (d) There doesn't exist a grammar that is LL(2) and LR(0)



For More Study Materials : www.keralanotes.com

45. Which of the following in incorrect?

(a) every LR(0) is CLR(1) (b) every LL(1) is LR(0)

(c) every LR(1) is CFG (d) every SRL(1) is CLR(1)

46. Consider the grammar show below

<life> → <session><session>

<session> → play <session>|rest

In the LR (0) machine for the grammar consider the items

<life> → <session><session>,

<session> → play. <session>, <session> → play< session>.

Choose the correct answer

(a) No two of the items occur in the same state

(b) All there of the items occur in same state

(c) At least two of them occur in multiple states

(d) None of the above

47. Consider the following grammar

<expression> → <factor><rest>

<rest> → *<expression> | ε

<factor> → identifier

In the LR(0) machine the no two of the following LR(0) items appear in the same state

(i) <expression> → <factor> <rest>

(ii) <expression> → *<rest>

(iii) <factor> → identifier

(a) (i) and (ii) (b) (i) and (iii)

(c) (ii) and (iii) (d) None of the above

48. S → SB / A

A → a

B → b

Find the closure set of $S^1$ → . S for the LR(0) machine is _____.

(a) $S^1$ → .S
S → .SB
S → . A
A → .a
B → .b
→

(b) $S^1$ → S
S → . SB
S → . A
A → .a



For More Study Materials : www.keralanotes.com

(c) $S^1$ .S (d) None

S ⟶ .SB

S ⟶ .A

49. S ⟶ AA

A ⟶ aA/b

In the LR(0) machine for the above grammar, the number of states are____.

50. The LR(0) parser for the grammar

E ⟶ E+T|T

T ⟶ T*F|F

F ⟶ id

Contains _____ number of conflicts.

51. Consider the grammar

<statement> ⟶ a <statement> | b

<statement > b|c

Choose the false statement

A ⟶ A + A|i

Choose the proper statement regarding the property of the above grammar.

(a) The grammar is LR(1) but not LALR(1)

(b) The grammar is LALR(1) but not LL(1)

(c) The grammar is SLR(1) but not LR(0)

(d) None of the above

53. Consider the grammar

$S^1$ ⟶ S, S ⟶ aAd|bBd|aBe|bAe, A ⟶ c, B ⟶ c

The grammar is

(a) LR(1) but not LALR(1) (b) LALR(1) but not LL(1)

(c) SLR(1) but not LR(0) (d) None of the above

54. Consider the grammar

S Aa|bAc|Bc|bBa, A ⟶ a, B ⟶ d

(a) The grammar I LR(1) but not LALR(1)

(b) LALR(1) but not LL(1)

(c) SLR(1) but not LR(0)

(d) None of the above



For More Study Materials : www.keralanotes.com

55. What is the conflict in LR(1) construction for the grammar

A → aA|Ab|d

(a) RR conflict (b) SR conflict

(c) Both RR and SR conflicts (d) No conflict

56. Consider the grammar

S → Aa|bAc|dc|bda

A → d

(a) The grammar is LL(1) (b) The grammar is LR(0)

(c) The grammar is SLR(1) (d) The grammar is LALR(1)

57. What is the closure of S → .A, $ for the grammar S A, A → AB/ε, B b →

(a) S → .A,$b
A → .AB$/b
A → .,$/b

(b) S → .A, $
A .AB,$
A → .,$
B → .b, $

(c) S → .A,$
A → .AB,$/b
A → .,$/b

(d) None

58. The grammar E → E + T/T; T → a is

(a) not LR(0) (b) not SLR(1) but CLR(1)

(c) CLR(1) (d) not LALR(1)

59. What is the conflict in LR(1) construction for the grammar S Sa/bS/ c is _____.

(a) R-R conflict (b) S-R conflict

(c) no conflict (d) both RR and SR conflict

60. S → L = R/R

L → * R/id
R → L
In the LR (1) items of the above grammar, the closure set of items in
$S^1$ → .S, $ contains
(a) L → *R, = (b) L → *R,$
(c) L → *R, = /$ (d) L → .id, =

61. Consider the grammars

1) S → L = R|R, R → L, L → *R|id
2) S → Aa|bAc|dc|dba, A → d

The grammar are
(a) LR(0) (b) LALR(1) but not SLR (1)
(c) SLR(1) but not LR(0) (d) None of the above



For More Study Materials : www.keralanotes.com

62. The grammar

S → (s)/ ε is

(a) LL (1) (b) LR(0)

(c) CLR (1) (d) both LL(1) and LR(1)

63. Consider the grammar

<statement > :: = [<statement >] |assignment | null

Let the Number of states in the SLR (1), LR(1) and LALR(1) parsers for the grammar be N1, N3 and N2 respectively. The following relationship holds good

(a) N1<N2>N3 (b) N1 = N2 > N3

(c) N1 = N2 = N3 (d) N1>=N3>=N2

64. Let there are '10' states for a grammar which is SLR(1), then the number of states in LALR(1) parser is _____.

65. The parser generator tool YACC uses ____ parsing table

(a) LR(0) (b) SLR (1)

(c) LALR (1) (d) LR (1)

**Key**

| | | | | | |
|---|---|---|---|---|---|
| 01. (b) | 02. (d) | 03. (a) | 04.(d) | 05. 2 | 06. (c) 07. (a) |
| 08. (c) | 09. (c) | 10. (c) | 11. (b) | 12. 144 | 13. (b) |
| 14. (c) | 15. (c) | 16. (d) | 17. (c) | 18. (d) | 19. (a) |
| 20. (c) | 21. (d) | 22. (d) | 23. (c) | 24. (c) | 25. (c) |
| 26. (a) | 27. (c) | 28. (d) | 29. (b) | 30. (d) | 31. (d) |
| 32. (c) | 33. (c) | 34. (d) | 35. (d) | 36. (d) | 37. (b) |
| 38. (b) | 39. (a) | 40. (d) | 41. (d) | 42. (d) | 43. (c) |
| 44. (d) | 45. (b) | 46. (a) | 47. (b) | 48. (b) | 49. 7 |
| 50.2 | 51.(a) | 52.(d) | 53. (a) | 54. (a) | 55.(b) |
| 56. (d) | 57. (c) | 58. (c) | 59. (b) | 60. (c) | 61. (a) |
| 62. (d) | 63. (b) | 64. 10 | 65. (c) | | |

keralanotes.com



For More Study Materials : www.keralanotes.com

# Syntax Directed

# Translation Scheme

**Concepts & Definitions**

- S- attributes – Synthesized attributes entered to bottom up parsing .
- Semantic Routines – A router attached to each rule of context free grammar for purposes of semantics.
- Semantic stacks- stacks parallel to the syntax stack for semantic routing.
- Display – A replication of the static chain for speeding up scope of variables.
- Dynamic Chain- the order in which procedure are called.
- Static chain – the static scoping of variables.
- Row major order – Storing an array row wise.
- Column Major order storing an array column wise

**4.1 Concept**

Syntax deals with form & Semantics with Meaning

**4.2 Concept**

The 'syntax of a HLL is specified by a context free grammar (Also known as the Backus Normal form or Backus Normal form or Backus Noun Form). The Syntax specification of a Hill yields a superset of the correct HLL Programs.

4.3 The Semantics of a HLL can be specified by context- sensitive language. but this will be a cumbersome specification with a lot of undecidability results.

**4.4 Concepts**

The Semantics of HLL are normally given in a natural language like English.

**4.5. Concept**

The Semantic specification is a superset of the programs specified by the Syntax specification and a proper subset of the set of correct programs.

**4.6. Concept**

The undecidability of the halting problem of Turing machine shows that semantic specifications only yields statically correct programs.

**4.7 Concept**

The concept of static is related to regular functions & the concept of dynamic to R.E sets

keralanotes.com



For More Study Materials : www.keralanotes.com

**4.8 Concept**

The language L = (ww/w .. ..( a+b)$^+$ } models the semantic requirement of having specification & declarations occurs before use. I is a context – sensitive feature as L is a CSL & cannot be expressed by a context free grammar.

**4.9. Concept**

The language L = ($a^i$ $b^j$ $a^i$ $b^j$ j $\geq$ 1 } is a context – sensitive language that is not context – free . It models the requirement that specification & use of function with relation to order and number of parameters match. It is a semantic requirement that is not context – free.

**4.10 Corollary**

Fortran is a context – sensitive language as are C, C ++ & Java .

**4.11 Definition**

Translation is made in the front end of a compiler to an intermediate code

**4.12 Definition**

Translation is made in the front end of a compiler to an intermediate code is not a must. Translation from the Syntax analysis phase using Semantic rules can directly yield the machine code. however the Semantic routines will become very bulky. To cut down the complexity of the Semantic routines & to facilities machine independent optimization, a translation to the intermediate code is preferred

**4.13 Varieties of intermediate code**

Let us take the code for the expression

a + b *c/b *c*f

i) Postfix notation

abc*b/c*f +

ii). Quadruples or 3 – address code

1. (#, b,c,$T_1$)
2. (/, $T_1$, b, $T_2$)
3. (*, $T_2$, c, $T_3$)
4. (*, $T_3$, f, $T_4$)
5. (+, a $T_4$, $T_5$)

iii) Triples for a +b ↑C* f

1. ( ↑, b, c) \
2. (*, (1), (1) )



For More Study Materials : www.keralanotes.com

3. (*, (2), f)

4. (+, a, (3) )

iv). Indirect triples (form the triples )

((1))

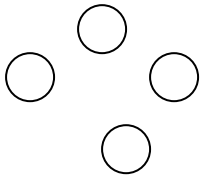
((2))

((3))

((4))

(v) **Trees**

Consider the expression a+b*c



vi). DAGs( directed Analysis Graph )

This is a representation that is usefull to isolate common sub-expressions. DAGs can be constructed using the same techniques as those used to construct Syntax trees.

A DAGs has atomic operands & interior nodes that correspond to operators just like a Syntax tree. However a node N in a DAG may have more that one parent if N is a common sub-expression . Thus a DAG represent expressions succinctly.

Example of DAG

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

……..

…….

4.14. Translation to intermediate 3- address code. A sample snipped of coe void quick sort (int m, int n ) /*recursively sort a (m) through a(n) */

int i, j ;

int, v, x;

if (n< =m )return;

/* fragrant begin here */

i= m – 1; j =n; v=a(n)

While (i)



For More Study Materials : www.keralanotes.com

```
do i= I+1 ; while (a[i]< v );
do j=j -1 ;  while (a[j]>v
if (i > =j ) break ;
x = a[i]; a(i)= a(j) = a[j] = x ; / * swap a(i), a(n) */
```

/* fragnant ends here */

quick sort (m.j); quick sort (i+ 1, n )*

The program fragment deals with the famous quick sort sorting algorithm . Here a sub- array A(M) TO A(N) is to be sorted. The algorithm proceeds by selected arbitratrarily apportion elements a(v) in the subarray. The elements are then arranged around the partition element, the losses over to the left & the greater one's to the right. this partitioning step is recursively applied to the left & right sub arrays using the Divide & Conquer strategy.

The translation of the fragrant to the intermediate code is discussed in detail.

- void quick sort (int m, int n )

  the function quick sort is only a procedure m & n are passed by value and define the subarray a(m). . . a(n) that is to be sorted. Let it be assumed that a is an integer array.

- int. I, j ;

  i is a pointer which moves left to right from m to n. It skips over all elements that are smaller than the pivot element. j is a pointer which moves right to left from n to n. It skips over all elements that are larger that the pivot element .

- in v, x ;

  a(v) is the pivot element for the partition step. It can be arbitrarily chosen. It is chosen. It is chosen here to be the rightmost elements of the partition. a[X] is temporary value used for swapping purposes .

if(n<=m )return;

If = m there is only one element in the range a(m) to a(n) & it is trivially sorted. If n< m then the partition a(n) contains null elements & is trivially sorted . this expresses the terminal condition for the Divide & Conquer step. If there is only 0 or I elements to be sorted then the subarray is trivially sorted.

- i=m-1;

Here the left pointer is set up.

j = n



For More Study Materials : www.keralanotes.com

Here the right pointer is set up.

- v=a(n)

Here the pivot is chose as the right most elements

- while (1)

Repeat the above loop continuously till the pointer cross .

- do i =i+1; while (a(i)< n

Advances the left pointer step by step by step to the right skipping over large /small elements.

- do j = j -1 ; while (a[j] > n );

  Advances the right pointer step by step to the left skipping over large elements.

- do (i>=j) break;

  The pointer meet or have crossed-return

  x = a(i);a(i)=a(j);a(j)= n ;

  swap a(i) over a(j)

- x= a(i); a(i)= a(n), a(n)= x;

  position the pivot

- quick sort (m.j );

  Sort left partition

- quick sort (I + I, n);

  Sort partition.

Sample BNF

< code snippet>; : = function definition >

< function definition > type> id (<param list>)

{ <declaration list >< statement list> }

< param list > =param> | <param>, < param list >

<param > : <:type> id | < expression list > }

<type > : : + int / char /float/ void

<expression list > :: = expression > |< expression >, < expression list >

<expression > :: = expression > |< loop >, < expression > < loop>

<expression>

|-<expression>

|(expression >)

| id



For More Study Materials : www.keralanotes.com

< rop > : : = </>/<=/>==/!

<loop>:: =+/-/*/ /

<declaration list > :: = declaration > |< declaration >, <declaration list >

<statement list > :: = <statement > |<statement >, < statement list >

<statement > :: = var> |< expression >, (< expression list >) <statement >

| do < statement > while (<expression> )

| while (<expression>) < statement

| for (<expression>; < expression>;<expression >)< statement >

| null

| continue

| break

| return

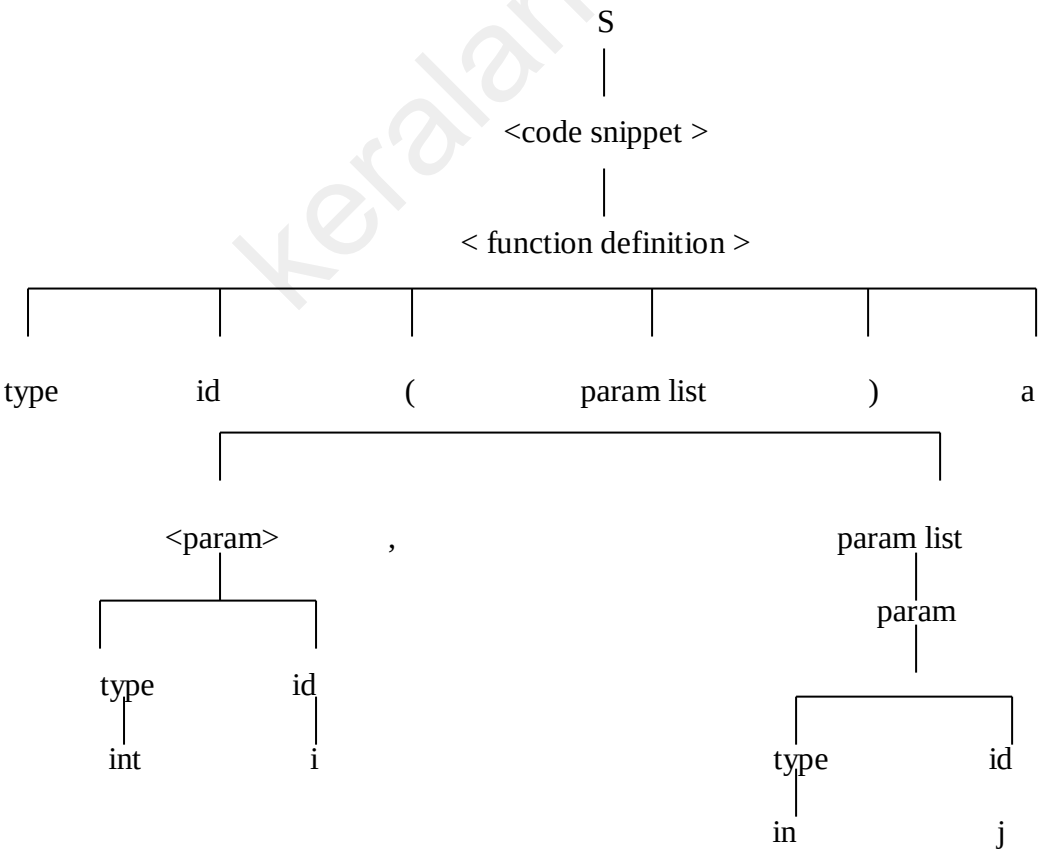
| < function call >

Var ; id/id [ <expression>]

<function call >:: id (<expression list ]

< declaration list>:: = < declration >/< declarataion list >< declaration

<declaration > ::<type> id:




keralanotes.com



For More Study Materials : www.keralanotes.com

Translating to the intermediate 3 – address code. We generate the intermediate code as we reduce the productions

1. <decl>:: int I

Set up symbol table contain i as a integer type identifies of with 4 bytes

2. <decl> :: = intj :

Set up symbol table to contain j as an integer type identifies of width 4 bytes.

3. <decl> :: = intv :

Set up symbol table to contain variable v as an integer type of width bytes

var = <expression > ;

<statement list >

/* code generated is

(14) $t_6$ = 4* i

(15)x= a[$t_6$ ]*/

while (1) { do < statement > while (exp>);

do <statement > while (<exp> );

if (< expression> break ;

var = <expression >

var = < expression> ;

< statement list>

/* code generated

(16)$t_7$ 4 * i

(17) $t_8$ = 4 * j

(18) $t_9$ = a ($t_8$)

(19)a [$t_7$]= $t_9$*/

⟹ while (1) {do <statement > while (<exp> );



For More Study Materials : www.keralanotes.com

do < statement > while (<exp>);

if (exp) break;

var = <expression > ;

var = <expression >;

var = < expression >; }

/* code generated

(20) $t_{10}$ = 4*j

(21) a [$t_{10}$] = x

(22) go to (8)

(23) . . . . . */

11. <statement> ⇨ var = expression >

⇨ X= a(i);

/* code generated

(23)$t_{11}$ = 4 * i

(24) x = a($t_{11}$)*/

12. <statement > ⇨ var = <expression>

a[i] = a(n)

/*code generated

(25) $t_{12}$ =4* i

(26) $t_{13}$ = 4*n

(27) $t_{14}$ = a($t_{13}$)

(28) a[$t_{12}$] = $t_{14}$ */

13. <statement >⇨ var = <expression >

⇨a(n)=x

(29) $t_{15}$ = 4*n

keralanotes.com



For More Study Materials : www.keralanotes.com

(30)a[$t_{15}$] =x

14. Putting the code together

| | |
|---|---|
| (1). i=m-1 | (2) j=n |
| (3) $t_1$=4*n | (4) v=a($t_1$) |
| (5) i= i + I | (6) $t_2$ = 4*I |
| (7) $t_3$=a($t_2$) | (8) if$t_3$< v go to (5) |
| (9) j=j-1 | (10) $t_4$ =4*j |
| (11) $t_5$=a ($t_4$) | (12) if $t_5$>v go to (9) |
| (13) if i > = goto | (14) $t_6$= 4*i |
| (15) x = a($t_8$) | (16) $t_7$=4*i |
| (17) $t_8$=4 * j | (18) $t_9$ =a($t_8$) |
| (19)a($t_7$)= $t_9$ | (20) $t_{10}$=4*j |
| (21) a($t_{10}$)=x | (22) go to (8) |
| (23) $t_{11}$=4*i | (24) x =a($t_{11}$) |
| (25) $t_{12}$ =4*I; | (26) $t_{13}$4*n |
| (27) $t_{14}$ =a($t_{13}$) | (28) a[$t_{12}$] =$t_{14}$ |
| (29) $t_{15}$=4*n | (30) a($t_{15}$)=x |

the flow graph

……..

……..

intermediate forms for control structure

1. S → if (B) $S_1$
2. S → if (B)$S_1$ else $S_2$
3. S → while (B)$S_1$
4. S → do S while (B) ;

keralanotes.com



For More Study Materials : www.keralanotes.com

5. S ⟶ for (E1; E2;E3)S

6. S ⟶ switch (E) {case ei:Si}default : $S_n$+1

01. The if statement

Construct : if (B) $S_1$

Code generated : B. code

$S_1$ . code

02. Modified construct : if $M_1$(B) $M_2S_1M_2$

Code generates $M_1$ ⟶ to B true

B. code

($M_2$) ⟶ to B. false

B. true: $S_1$ code

B. false ; ($M_3$)\

Ex: if (a >b ) then c=d+e

t = a > b

IF J (t, false)

c= d+e

false : nop

03. The if – else statement

Construct : if (B) $S_3$ else $S_4$

Code generated : B. code

$S_3$. code

$S_4$. Code

04. Modified construct

If $M_4$(B) ⟶ to B. true

B. code

keralanotes.com



For More Study Materials : www.keralanotes.com

$M_5 \longrightarrow$ to B false

B. true : $S_3$- code

$M_6 \longrightarrow$ to exit

B. false : $S_4$

Exit : NOP

05. Construct : while (B) $S_5$

Code generated : B. code

$S_5$.Code

06. Modified construct

$M_8$ while $M_9$ (B) $M_{10}S_5M_{11}$

Start ($M_8$) $\longrightarrow$ B. true

($M_9$)

B. code

$M_{10} \longrightarrow$ B. false

$S_5$. code

$M_{11} \longrightarrow$ goto start

B. false : NOP

07. Construct : S $\longrightarrow$ do S while (B);

Code generated : S code

B. Code

08. Modified construct

S $\longrightarrow M_{12}$ do S while $M_{13}$(B)$M_{14}$

Code generated

Start : $M_{12}$

S. code

$M_{13} \longrightarrow$ B. true

B. code

$M_{14} \longrightarrow$ go to B. false

got to start

B. false : NOP

09. Construct S $\longrightarrow$ for (E1;E2;E3;)S

keralanotes.com



For More Study Materials : www.keralanotes.com

Code

E1. code

E2. code

E3. code

S. code

10. Modified construct

S $\longrightarrow$ for ($M_1E_1$;$M_2E_3$) $M_4SM_5$

Flow chart for

…….

…….

……..

Code generated

Start : $M_1$

E1. code

$M_2 \longrightarrow$ go t $E_2$ true

Start $E_2$ : $E_2$ Code

$M_3 \longrightarrow E_2$ false

go to start S;

Start S : $S_4$ code

go to start $E_3$

The SDTS for control structures

S $\longrightarrow$ if ($B_1$) $M_1$ $S_1$ $M_2$

$M_1 \longrightarrow_E$ {gen(if j, $B_2$(if j, $B_1$val,-); $M_1$ Value, codeptr )}

$M_2 \longrightarrow_E$ (backpatch ($M_1$value, codeptr)}

S $\longrightarrow$ if ($B_2$)$M_3S_2$ else $M_4S_3M_5$

$M_3 \longrightarrow_E$ (gen(if j, $B_2$ val,-); $M_3$ value = codeptr

$M_4 \longrightarrow$ {gen (UJ,-); $M_4$. value = codeptr ;

backpatch ($M_3$.value, condeptr )}

$M_5 \longrightarrow_E$ {backpatch ($M_4$. value, codeptr);}

S $\longrightarrow M_6$ while ($B_3$) $M_7S_4M_8$

$M_6 \longrightarrow_E$ {$M_6$value =codeptr ;}

$M_7 \longrightarrow_E$ gen(if j,$B_3$. val,-); $M_7$value =codepr ;}

$M_8 \longrightarrow_E$ {gen(UJ, $M_6$.val) backpatch ($M_7$val, codeptr);}



For More Study Materials : www.keralanotes.com

S $\rightarrow$ $M_9$ do S while $M_{10}$(B)$M_{11}$

$M_9 \rightarrow$ $_E${$M_9$val = codeptr;}

$M_{10} \rightarrow$ $_E$

$M_{11} \rightarrow$ $_E${ge = codeptr;}

Syntax Directed Translation Scheme for evaluation of expressions.

E1 $\rightarrow$ E#

E $\rightarrow$ E + E

E $\rightarrow$ E*E

E $\rightarrow$ (E)

E $\rightarrow$ num

Sample expression

4+3*6+4#

The derivation

E1 $\Rightarrow$ E #

$\Rightarrow$ E+E

$\Rightarrow$ E+E+4

$\Rightarrow$ E+E*E+4

$\Rightarrow$ E+E*6+4

$\Rightarrow$ E+3*6+4

$\Rightarrow$ 4+3*6+4

The SDTS scheme

$E_1 \rightarrow$ E#{$E_1$ value=E. Value }

E $\rightarrow$ $E_1$ +$E_2${E. Value = $E_1$value + $E_2$ value }

E $\rightarrow$ $E_1$* $E_2$ (E value = $E_1$value +$E_2$.value

E $\rightarrow$ {$E_1$} {E. value =$E_1$value }

E $\rightarrow$ num {E. value = num}

The derivation tree

……

……..

…….

keralanotes.com



For More Study Materials : www.keralanotes.com

The productions used in the reverse of a rightmost derivation.

$E_1 \longrightarrow 4$

$E_2 \longrightarrow 3$

$E_3 \longrightarrow 6$

$E_4 \longrightarrow E_2*E_3=18$

$E_5 \longrightarrow 4$

$E_6 \longrightarrow E_4+E_5=22$

$E_7 \longrightarrow E_1E_6=26$

The tree with reductions

Example 4.1:

Write SDT to build a syntax tree for expression ?

Sol : $E_1 \longrightarrow E\#$

$E \quad E_1 * E_2$

$(E_1)$

Id

The SDT

$E \longrightarrow$ id {create E tree (1ptr =rptr=null, val=0)}

$E \longrightarrow E_1+E_2$ {E. tree = (1ptr = $E_1$tree , rptr= $E_2$.tree, val=$E_1$.val+val}

$E \longrightarrow E_1*E_2${E.tree=1ptr=$E_1$.tree, rptr=$E_2$.tree, val =$E_1$val*$E_2$val}

$E \longrightarrow (E_1)$ {E.tree= $E_1$tree}

Note : Intr is left pointer and rptr and is right pointer

Example : 4:1:

Write SDT to build a syntax tree a syntax tree for expression ?

Sol : E1 $\longrightarrow$E #

$E \longrightarrow E_1+E_2$

$|E_1*E_2$

$| (E_v)$

| id

The SDT

$E \longrightarrow$ id (create E. tree (1ptr = rptr=null, val=0)}

$E \longrightarrow E_1+E_2$ {E.tree =(1ptr = $E_1$tree, rptr=$E_2$ tree, val =$E_1$. val +$E_2$. val}

$E \longrightarrow E_1*E2$(E.tree =1ptr=$E_1$.tree, rptr=E2.tree val= $E_1$ val*$E_2$val }

$E \longrightarrow (E_1)$ {E.tree =$E_1$tree)

Note : 1ptr is left pointer and rptr is right pointer



For More Study Materials : www.keralanotes.com

Example 4.2:

Write an SDT to create DAG ?

Sol : E → id {If id does not exist then

Get E.tree (1ptr = rptr = null, val=0)}

E → $E_1$+$E_2$ {E.tree= (1ptr = $E_1$.tree, val = $E_1$. val +$E_2$.val)}

E → $E_1$*$E_2$ {E.tree =(1ptr= $E_1$. tree, rptr= $E_2$tree, val*$E_2$. val)}

E → ($E_1$) { E.tree = $E_1$.tree }

Example 4:3

Write SDT to store type information in the symbol table ?

Sol : D → T(Ln = T: type }

L

T → int {T.type = integers}

T → real{T. type = real }

L → {$L_1$.in : $L_{in}$)

$L_1$id{add type (id. Entry, Lin)}

L id{add type(id. Entry , L.in )}

Note : id entry is the entry is the entry in the table

Write an SDT to generate three address code for any expression ?

Sol : E1 → E#

E → $E_1$+$T_1$ { new t ;t = out(+,$E_1$t. T.t ); E.t=t}

E → $T_1$ {E.t = $T_1$t}

$T_1$ → $T_1$*$F_1$(new t; t= out(*,T,t,$F_1$t); T.t = t}

T → $F_1$ {T. t=$F_1$.t}

F → id {F.t =id. name }

Note : new t creates a new temporary

Example 4:5:

Write an SDT to issue error messages ?

Sol : $E_1$ → E#

E → $E_1$+$E_2$

/$E_1$*$E_2$

/(E1)

/id

Both operands have to be of type float or int

$E_1$ → E#

E → $E_1$+$E_2$ {if ($E_1$type =int & $E_2$. type = float )



For More Study Materials : www.keralanotes.com

Error() ;

Else if ($E_1$. type = float & E2.type = int)

Error ( );

Similarly for $E_1$*$E_2$

**Example 4.6:**

Write an SDT to count the number of terminals.

Sol: $E_1 \rightarrow$ E # ($E_1$. count = E. count + 1 }

E $\rightarrow$ $E_1$+T (E. count =$E_1$count +T. count }

E $\rightarrow$ T(E. count =T.count}

T $\rightarrow$ $T_1$*F{T. count = $T_1$count +F. count+1}

T $\rightarrow$ F {T.count = F.count}

F $\rightarrow$ (E){F.count = E.count }

F $\rightarrow$ id (F. count = 1}

**Example : 4.7**

Write an SDT to reverse an expression

Sol : E1 $\rightarrow$ E# (E1. val= # |E.val }

E $\rightarrow$ $E_1$+T {E.val = || '+'|| $E_1$ val

E $\rightarrow$ T { E. val=T. Val}

T $\rightarrow$ $T_1$ * F {. val || '*'|| $T_1$val}

T $\rightarrow$ F {T.val=F. val }

F $\rightarrow$ (E) {F.val = ')| | E. val | | "(''}

F $\rightarrow$ id {F.val = id name }

Example 4.8 :

Write an SDT for converting a binary number to its decimal equivalent.

Sol : N $\rightarrow$ ND/D

D $\rightarrow$ 0/1

The SDTS

D $\rightarrow$ 0 {(D. decval = 0)}

D $\rightarrow$ 1 {D. decval = 1)}

N $\rightarrow$ D (N.decval =D.decval }

N $\rightarrow$ $N_1$D(N.decval = $N_1$decval*2+D.decval }

Note : 'decval' is decimal value

Example 4.9 :

Write an ADT for converting a integer string "1234" to its decimal equivalent



For More Study Materials : www.keralanotes.com

Sol : string ⟶ String chair string char /char

Char ⟶ 0' – '9'

SDTS

Char ⟶ '0'- '9'{ char . val = char =0 to 9 }

String ⟶ char {String. Val = char. Val }

String ⟶ $string_1$ char { string. Val = string. Val*10+char. Val}

Example 4.10

Write postix SDT to store type information in symbol table

Sol: $E_1$ ⟶ E#

E ⟶ $E_1$+T {out ('+')}

E ⟶ T

T ⟶ $T_1$*F {out(*)}

T ⟶ F

F ⟶ (E)

F ⟶ id (enter symbol table id. Name = id. Type }

**Classroom Practice Questions**

01. Syntax Directed translation is part of

(a) Lexical Analysis (b) Syntax Analysis

(c) Semantic Analysis (d) Intermediate code generation

02. The following SDT is

E ⟶ $E_1$+T {E val =$E_1$+ T. Val}

E ⟶ T {E val =$E_1$+ T. Val}

T ⟶ id {T. Val= id }

a) S-attributed b) L-attributed

c) Both (a)and (b) (d) None

03. The following SDT checks

S ⟶ aSb { S. count =S. count +2}

S ⟶ bSa {S. count = S. count +2}

S ⟶ $_E$ { S. count =0}

(a) Equal number of a's and b's.



For More Study Materials : www.keralanotes.com

(b) Number of a 's or number of b's in given string

(c) Number of a's and b's in a given string

(d) None

04. Consider the following Syntax Direct Translation scheme

E ⟶ E+T { E. value = '+' || E. Value || T. value )

E ⟶ E –T {E. value = '-' || E. value || T. value )

E ⟶ T{E. value =T. value }

T ⟶ id {T. value= id} (Assume || is string concatenation )

The above SDTS specifies _____

a) Expression Evaluation b) Infix to postfix conversion

b) Infix to prefix conversion d) None of the Above

05. Consider the Syntax Directed Definition below

S ⟶ $S_1$ $S_2$c {S, t=$S_1$, t-$S_2$-t}

S ⟶ a { S, t=2}

S ⟶ b {S, t=6}

If the Top down parser carries out the translation for an input string "babcc" output is _____

a) 0 (b) 6

c) 10 (d) 5

06. Consider the SDT given below,

S ⟶ $S_1$ $S_2$c { S, val* $S_2$, Val=4}

S ⟶ a {S, val=6}

S ⟶ b {S, val=2 }

If the SDT gives the input string abc, then the output is

(a) 5 (`b) 8

(c) 0 (d) 2

07. Consider the SDTS (using synthesized attributes

$E_1$ ⟶ E2 *E3 ($E_1$ sem= $E_2$.sem x E3.sem]

$E_4$ ⟶ $E_5$+$E_6$ [$E_4$sem =$E_5$.sem +$E_6$.sem ]

E ⟶ id [E.sem =1]

The output for the input a+b+c+d+e is



For More Study Materials : www.keralanotes.com

a) 3 (b) 4 (c) 5 (d) 6

08. Consider the SDTS (using synthesized attributes )

S → aS out(''1'')

S → a out("2")

The output for the input aaaa is

a). "1111" b) "2222"

c). "2111" d) "1112''

09. Consider the SDTS (using synthesized attributes)

S → aA out ("1")

S → aa out ("2'')

A → a out ("3'')

The output for the input aa is

a) 31 or 2 b) 2

b) 31 d) None of the above

10. Consider the SDTS (using synthesized attributes )

S → aB {out "cat''}

S → bA {out "dog"}

B → aBB {out "pig}

A → bAA {out "cow''}

B → b {out "deer"}

A a (out " lion" }

The output for aabb is

a) deer deer pig cat b) cat dog pig lion

c) cat cow deer lion c) cat deer dog pig

11. Consider the following Syntax Directed Translation Scheme (SDTS), with non-terminals { S,A } and terminals {a, b}

S → aA{print 1 }

S → a{print 2 }

A → Sb(print 3}

Using the above SDTS, The output printed by a bottom-up parser , for the input aab is ;

(a) 1 3 2 b) 2 2 3



For More Study Materials : www.keralanotes.com

c) 2 3 1 d) syntax error

12. Here is a postfix SDT :

S ⟶ aS{print "X"}

S ⟶ bS {print "y"}

S ⟶ a {print "z"

S ⟶ b {print "z"

Suppose we execute this SDT in connection with a bottom-up parser. What will be printed in response to the input " ababb"

a) zxyxy b) zyxyy

c) zyxyx d) zyyxx

13. The output for the SDT

A⟶ A + A{print 1}

A⟶ a { print 2}

For a = a is

a) 22121 b) 21221

c) 21212 d) 12122

14. Consider the SDTS given below

E1 ⟶ E print ("S'')

E ⟶ E+E printf ("+"

E ⟶ E*E printf("*")

E ⟶(E)

E ⟶id print(id. Value)

If the we assume a bottom up shift reduce parser, the output for the input string a+b* c is

(a) abc *+

(b) ab+ c *

(c) abc*+or ab+c* depending on the compiler

(d) There will be an error and no parsing can take place

15. Consider the following translation scheme

S ⟶AB

B ⟶* A { print*} $B/_E$

A ⟶ C+A { print+} /C

C ⟶ S/id {print id }

keralanotes.com



For More Study Materials : www.keralanotes.com

Here id represents an integer For input '5* 6+7' this translation prints _____

(a) 3 7 (b) 5*6+7

(e)56*7+ (d) 5 6 7 +*

16. Consider the SDT given below

E → E ↑E { print }

E → E*E {print*}

E → id { print id }

If the shift reduce parser construct a parse tree for the input sentence a*b c, the ↑above translation prints.

(a) ab *c ↑ (b) a b c ↑*

© Both (d) None

……

……..

keralanotes.com



For More Study Materials : www.keralanotes.com

# 5. Intermediated Code Generation

**Introduction**

In the analysis –synthesis model of a compiler, the front end analyzes a source program and creates an intermediate representation from which the back end generates target code. Ideally, details of the source language are confined to the front end, and details of the target machine to the back end. With a suitably defined intermediate representation, a compiler for language I and machine j can then be built by combining the front end for language I with the back end for machine j. We assume that a compiler front end is organized as in figure 5.1 where parsing, static checking and intermediate –code generation are done sequentially, sometimes they can be combined and folded into parsing.

……

……

Static checking includes type checking, which ensues that operators are applied to compatible operands. It also includes any syntactic checks that remain after parsing.

In the process of translating a program in a given source language into code for a given target machine, a compiler may construct a sequence of intermediate representation. High –level representations are close to the source language and low-level representations are close to the source language and low-level representations are close to the target machine.

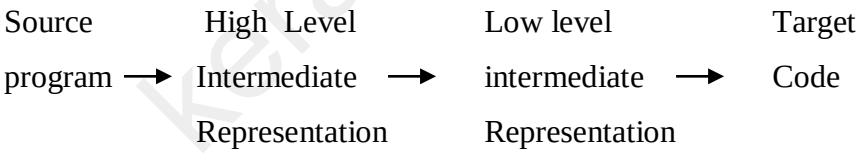



Fig 5.2 : A compiler might use a sequence of intermediate representations

**Three –Address Code**

In three –address Code, there is at the most one operator on the right side of an instruction : that is, no built –up arithmetic expressions are permitted. Thus a source-language expression like x +y*z might be translated into the sequence of three-address instructions

$T_1 = y*z$

$T_2 = x + t_1$

Where $t_1$ and $t_2$ are compiler – generated temporary names.

Example : three address code is linearized representation of a syntax tree or a DAG, in which explicit names correspond to the interior nodes of the graph.



For More Study Materials : www.keralanotes.com

…….

……..

Fig : 5.3 DAG and its corresponding three-address code

Quadruples

A quadruple (or just "quad") has four fields, which we call op, $arg_1$ $arg_2$ and result. The op field contains an internal code for the operator For instance, the three-address instruction x = y+z is represented by placing +in op, y in $arg_1$ z in $arg_2$ and x in result.

The following are some exceptions to this rule :

1. Instructions with unary operator like x = minus y or x = y do not use $arg_2$. Note that for a copy statement like x=y, op is = while for most other operations, the assignment operator is implied.
2. Operators like param use neither $arg_2$nor result.
3. Conditional unconditional jumps put the target label in result.

Example : Three –address code for the assignment a=b*-c+b*-c:

$T_1$= minus c

T2 = b*$T_1$

$T_3$ = minus c

$T_4$ = b* $t_3$

$T_5$= $t_2$+$t_4$

a = $t_3$

(a) Three-address code

| | OP | $Arg_1$ | $Arg_2$ | Result |
|---|---|---|---|---|
| | Minus | c | | $T_1$ |
| 1 | * | c | $T_1$ | $T_3$ |
| 2 | minus | c | | $T_3$ |
| 3 | * | b | $T_3$ | T |
| 4 | + | $t_2$ | $T_4$ | $T_5$ |
| 5 | = | $t_5$ | | a |

(b) Quadruples

Fig . Three-address code and its quadruple representation



For More Study Materials : www.keralanotes.com

**Triples**

A triple has only three fields, which we call op, $arg_1$ and $arg_2$. Using triples, we refer to the result of an operation x op y by its position, rather than by an explicit temporary name.

The DAG and triple representations of expressions are e equivalent.

……..

| | | | |
|---|---|---|---|
| 0 | minus | c | |
| 1 | * | b | (0) |
| 2 | minus | c | |
| 3 | * | b | (2) |
| 4 | + | (1) | (3) |
| 5 | = | a | (4) |

Fig 5.4: Representations of a=b*-c + b*c;

**Translation of Expressions**

The translation of expressions into three –address code. An expression with more than one operator, like a+b*c, will translate into instructions with at the most one operator per instruction. an array reference A(i) (j) will expand into a sequence of three-address instructions that calculate an address for reference

| Production | Semantic Rules |
|---|---|
| S ⟶ id =E; | S. code = E. code \|\| gen(top. get(id. lexeme)'=' E.addr) |
| E ⟶ $E_1$+$E_2$ | E.addr= new Temp()<br>E.code = $E_1$. code \|\| $E_2$. code \|\|<br>gen (E. addr '='$E_1$.addr' + $E_2$.addr) |
| 1-$E_1$ | E.addr=new Temp( )<br>E.code = $E_1$. code \|\|<br>Gen(E.addr '='minus' $E_1$addr) |
| \| ($E_1$) | E.addr=$E_1$addr<br>E.code =$E_1$code |
| \| id | E.addr= top.get(id.lexeme )<br><br>E.code = |

Fig 5.5: three-address code for expressions



For More Study Materials : www.keralanotes.com

Control Flow

The translation of statements such as if –else- statements and while statements is tied to the translation of Boolean expressions. In Programming languages, Boolean expressions are often used to

1. Alter the flow of control. Boolean expressions are used as conditional expressions is implicit in a position reached in a program. For example, in if (E) s, the expression E must be true if statement S in reached

2. Compute logical values. A Boolean expression can represent true or false as values. Such Boolean expressions can be evaluated in analogy to arithmetic expressions using three-address instructions with logical operations. as values. Such Boolean expressions can be evaluated in analogy to arithmetic expressions using three-address instructions with logical operations.

Flow-of control statements

The translation of Boolean expression into three-address code in the context of statements such as those generated by the following grammar:

S $\longrightarrow$ if (B)$S_1$

S $\longrightarrow$if (B)S1else $S_2$

S $\longrightarrow$while (B)$S_1$

In these productions, non-terminal B represents a Boolean expression and non-terminal S represents a Statement.

The translation of if (B) $S_1$consist of B. code followed by $S_1$. code , as illustrated in fig. Within B. code are jumps based on the value of B. If B is true, control flows to the first instruction of $S_1$ Code , and if B is false, control flows to the instruction immediately following $S_1$ code.

…….

…….

The labels for the jumps in B. code and S.code are managed using inherited attributes. With a Boolean expression B, we associate two labels: B. true, the label to which control flows if B is true, B. false, the label to which control flows if B if false. With a statement S we



For More Study Materials : www.keralanotes.com

associated an inherited attribute S. next denoting a label for the instruction immediately after the code for S. In some cases, the instruction immediately following S. code is a jump to some label L. A ump to a jump to L from withing S.code is avoided using S. next.

The syntax directed definition in fig. produces three-address code for Boolean expressions in the context of if, if else and while-statements

| Prduction | Semantic Rules |
|---|---|
| P S | S. next= newlabel ( )<br>P.code= S. code \|\| label(S. next) |
| S assign | S. code = assign. code |
| S if (B)$S_1$ | B.true=newlabel( )<br>B. false =$S_1$next = S. next<br>S.code =B.code\|\| label (B.true)<br>\|\|$S_1$code |
| S ⟶ if (B)$S_1$ else $S_2$ | B. true =newlabel( )<br>B. false = newlabel( )<br>$S_1$next = $S_{2\text{next = S. next}}$<br>S.code = B.code<br>\|\| label(B.true) \|\|$S_1$code<br>\|\| gen ('goto' S. next )<br>\|\| label (B. false)\|\|$S_2$ code |
| S While (B) $S_1$ | Begin = newlabel ( )<br>B.true = newlabel ( )<br>B. false = S. next<br>$S_1$next = begin<br>S. code = label ( begin) \|\| B. code<br>\|\| label(B.true)\|\| $S_1$.code<br>\|\| gen ('goto' begin) |
| S ⟶ $S_1$ $S_2$ | $S_1$next = newlabel ( )<br>$S_2$next =S. next<br>S. code= $S_{1.}$ code \|\| label($S_1$next) \|\| $S_{2.}$ code |



For More Study Materials : www.keralanotes.com

Fig 5.7: Syntax- directed definition for flow-of-control statements

**Control-Flow Translation of Boolean Expressions**

The semantic rules for Boolean expressions in Figure complement the semantic rules for statements in figure. A Boolean expression B is translated into three-address instructions that evaluate B using conditional and unconditional jumps to one of two labels: B true if B is tru, and B. false if B is false

| Production | Semantic Rules |
|---|---|
| B $\rightarrow$ $b_1$ \|\|$B_2$ | $B_1$true = B.true<br>B1.false =newlabel ( )<br>$B_2$ true = B. true<br>$B_2$ false = B. false<br>B. code = $B_{1.}$ code \|\| label ($B_{1.}$ false) \|\| $B_2$.code |
| B $\rightarrow$ $B_1$ & &$B_2$ | $B_1$.true = newlabel ( )<br>$B_1$ false = B. false<br>$B_2$ true = B. true<br>$B_2$ false = B. false<br>B. code = $B_1$code \|\|($B_1$. true ) \|\| $B_2$.code |
| B ! $B_1$ | $B_1$ true = B. false<br>$B_1$ false = B. true<br>B. code = $B_1$. code |
| B $\rightarrow$ $E_1$ rel $E_2$ | B.code = $E_{1.}$ code \|\| $E_2$ code<br>\|\| gen('if $E_1$. addr rel. op $E_2$addr 'goto' B. true)<br>\|\| gen ('goto' B. true ) |
| B true | B. code = gen ('goto' B.true ) |
| B false | B. code = gen ('goto' B. false) |

Fig 5.8 Generating three-address code for Booleans

Translate statement to 3 address statements

Example 5.1:

a=b+c*d/c

Sol : (1) (*, c,d)

(2) (/,(1),c)

(3) ( +, b,(2))



For More Study Materials : www.keralanotes.com

(4) (MOV, (3),9,-1)

Example 5.2 :

for (j=0:j<10; j++) a= b+c

Sol: The control flow :

……

……

The translation

……

……

The code

(1) (MOV, #0, j,-)

(2) (<, j#, 10, T )

(3) (IFJ, T, (8))

(4) UJ, (6)

(5) (+j, I, j)

(6) (+, b, c,a)

(7) UJ (2)

(8) NOP

Example 5.3:

While (A<C) and (B.D) do if A = 1 then C=C+1

else while A < = D do A=A+3

Sol : (1) (<A,C,T1)

(2) (>B,D, T2)

(3) (and, T1, T2, T)

(4) (IFJ, T, (15)

(6) (=,A,1, T1)

(7) (IFJ, T1,(10))

(8) (+, I,C,C )

(9) (UJ, (15) )

(10) (C=, A,D,T3)

(11) (IFJ, T3, (15)

(12) (+,#3, A A)

(13) (UJ, (10)-)

(14) (UJ, (1),-1)

keralanotes.com



For More Study Materials : www.keralanotes.com

(15) NOP

Example : 5.4

I f (a>7) or (b = = 5) then x =7 else y=z

Sol : (1) (>, 9,7,T1)

(2) ( = =,b,5,T2)

(3) (or, T1, T2,T )

(4) (IF J, T,(7)

(5) (MOV, #7, X, -)

(6) (UJ, (8))

(7) (MOV, #2, y-

(8) NOP

Example 5.5:

Switch (i)

Case 1 : a=10: break ;

Case 2; a=20; break;

Default : a= 50: break

Sol : The flow chart

……

…….

(1) (=,I,1,T1)

(2) (=,I, 2, T2)

(3) UJT1

(4) UJT2

(5) UJT

(6) (T1) (MOV, #,10,a)

(7) UJ(12)

(8) (T2)( MOV, # 20,a )

(9) UJ(12)

(10) (T3)MOV,#50,a)

(11) UJ(12)

(12) NOP

Example 5.6

-a+b/x↑d ↑ e * f/g

keralanotes.com



For More Study Materials : www.keralanotes.com

Sol (1) (-u, a, T )

(2) ↑, d,e,T2)

(3) ↑, x,T2,T3)

(4) (/,b, T3,T4)

(5) *, T4, 1, T5)

(6) /,T5,9,T6)

(7) (+,T1,T6,T7)

Note : '-u'is unary minus

Example : 5.7:

a<b or c<d and c < f

Sol (1) (<,a,b,T1)

(2) (<,c,d,T2)

(3) (<,c,f.T3)

(4) (and, T2,T3,T4)

(5) (or, T1, T4, T5)

**Directed Acyclic Graphs for Expressions**

A directed acyclic graph (DAG) for an expression identifies the common sub expressions in the expression. Like a syntax tree, a DAG has a node for every sub expression of the expression; an interior node represents an operator and its children represent its operands. The difference is that a node in a DAG representing a common sub expression has more than one "parent;" in a syntax tree.

example : Construct a DAG for the following expression

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

The leaf for a has two parents because a is common to the two sub expressions a and a*(b-c). Likewise, both occurrences of the common sub expression b-c are represented by the same node , which also has two parents.

……..

……..

Fig : 5.9: DAG for the expression a+a*b-c)+b-c)*d.

Classroom Practice Questions

01. One of the purpose of using intermediate code in compilers is to

a) make parsing and semantic analysis simpler



For More Study Materials : www.keralanotes.com

(b) improve error recovery and error reporting

(c) Increase the chances of reusing the machine independent code optimizes in the other compliers

(d) improve the register allocation

02. The following are not standard intermediate forms of the source program

(a) quadruples (b) triples

(c) postfix (d) machine language code

03. Which of the following is **not** a Three Address Code (TAC)

(a) return 0 (b) if a > b goto c

(c) a[i]=b., (d) none

04. Which of the following is **not** a form of Three Address Code (TAC) implementation?

(a) Quadruples (b) Triples

(c) Indirect triples (d) Syntax tree

05. The quadruple from of a+b*c-d is

(a) (1) (+,a,b, $T_1$)
(2) (*, $T_1$, c, $T_2$
(3) (-, T2, d, T3)

(b) (1) (*,b,c,$T_1$)
(b) (2) (+,a,$T_1$, $T_2$)
(3) (-, T2, d, $T_3$)

(c) (1) (+, a, b, $T_1$)
(2) (-, c, d, $T_2$)
(3) (*, $T_1$, $T_2$, $T_3$)

(d) None

06. Which of the following is in the quadruple from for the expression a or b and c or d ( Assume 'and' has higher precedence than 'or') Assume each quadruple is numbered from 1.

(a) (1) (0r, a, b, $T_1$) (b) (2) (and, b, c, $T_1$)

(c) (2) (or, a, $T_1$, $T_2$) (d) None

07. The indirect triples for (a*b)+(a*b*c) are

(a) (*, a, b)

(b) (1) (*, a, b)
(2) (*, a, b)
(3) (*, (2), c)
(4) (*, (1), (3))

(c) (1) (*, z, b)
(2) (*, (1), c)
(3) (+, (1), (2))

(d) None



For More Study Materials : www.keralanotes.com

08. Consider the following code segment

x = u – t.,

y = x * v.,

x = y + w.,

y = t – z.,

y = x * y .,

The minimum number of total variables required to convert the above code segment to static single assignment from is _____

09. Consider the following intermediate program in three address code

p = a – b

q = p * c

p = u * v

q = p + q

Which one of the following corresponds to a static single assignment from of the above code ?

(a)
$p_1$ = a – b
$q_1$ – $p_1$ * c
$p_1$ = u * v
$q_1$ = $p_1$ + $q_1$

(b)
$p_3$ = a – b
$q_4$ = $p_3$ * c
$p_4$ = u * v
$q_5$ = $p_4$ + $q_4$

(c)
$p_1$ = a – b
$q_1$ = $p_2$ * c
$p_3$ = u * v
$q_2$ = $p_4$ + $q_3$

(d)
$p_1$ = a -b
$q_1$ = p * c
$p_2$ = u * v
$q_2$ = p + q

10. Choose the incorrect statement

(a) DAG representation of a basic block enables elimination of common sub expression.

(b) DAG representation of a basic block enables dead code elimination.

(c) DAG representation allows optical register usage by specifying & cotrolling the lifetime of variables in registers.

(d) None

11. The DAG for a*b*b is

…….

……

12. The DAG for the expression



For More Study Materials : www.keralanotes.com

13. Consider the Three Address code below

d = a + b

e = c + d

f = d + e

b = c + e

e = b + f

The number of nodes in the DAG constructed for the above block of statements are _____

14. Consider the following block of three address statements

a= c + d

e= a + d

d = c + d

Then the number of nodes in the DAG constructed for the above block of statement is

15. For the code below

a = b + c

c = a + d

d = b + c

e = d – b

a = e + b

obtain the minimum number of registers to be allocated by the graph coloring algorithm .

(a) 2 (b) 3

(c) 4 (d) 5

16. Consider the basic block given below

a = b + c

c = a + d

d = b + c

e = d – b

a = e + b

The minimum number of nodes and edges present in the DAG representation of the above basic block respectively.

(a) 6 and 6 (b) 8 and 10

(c) 9 and 12 (d) 4 and 4

keralanotes.com



For More Study Materials : www.keralanotes.com

17. For a C program accessing X (i) (j) (k), the following intermediate code is generated by a complier. Assume that the size of an integer is 32 bits and the size of character is 8 bits.

t0 = i*1024

t1 = j*32

t2 = k*4

t3 = t1+t0

t4 = t3+t2

t5 = X(t4)

Which one of the following statements about the source code for the C program is CORRECT ?

a) X is declared as "int X[32] [32] [8]"

b) X is declared as " int X [4] [1024] [32]"

c) X is declared as "char X [4] [32] [8]"

d) X is declared as "char X[32] [16] [2]

18. The following syntax tree :

……

……..

Represent the assignment

a = (b-(c+d) + e*f;

Convert this tree to (op, arg1, arg2)triples, following these rules :

1. Evaluate the right sub tree of anode before the left subtree.
2. Number the instruction (1), (2) .

Then identify the list below , the one triple, with its instruction number, that would appear in your translation

(a) (1) [+, e,f ]

b) (3)[*,e,f]

(c) (3) [-, b, (2)

d) (4) [ +,b(1)

Key

01.(c) 02(d) 03(d) 04(d) 04(d) 05(b) 06(c) 07(c)

09 (b) 10 (d) 11 (d) 12(b) 13.8 14.4 15.(b) 16(b)

17 (a) 18(b)



For More Study Materials : www.keralanotes.com

# Chapter – 1

## Finite Automata and Regular Sets

### Level – 1 Questions

01. The class of regular sets are

(a) closed under finite union

(b) sometimes closed under infinite union

(c) closed under finite intersection

(d) all of the above

02. When and NFA is converted to an equivalent DFA accepting the same language, the number of states

(a) necessarily decreases

(b) necessarily increases

(c) always remains the same

(d) sometimes remains the same

03. Which of the following is most approximate in the case of Finite language

(a) It can be regular language

(b) It can be CFL

(c) It can be CSL

(d) It can be REL

04. Choose the correct answer

(a) The class of finite automata can be put into a one to one correspondence with the integers.

(b) The class of finite automata can be put into a one to one correspondence with the formal languages.

(c) The class of push down automata is larger than the class of finite automata and can be put into a one to one correspondence.

(d) None of the above

05. The minimal finite automata accepting the set of all strings in $(0+1)^*$ that has the last two symbols same has

(a) 40 states (b) 5 states

(c) 1000 states (d) 1000000 states

keralanotes.com

06. The set of all strings over {0, 1} where the $10^{th}$ symbol from the right end is a 1 and the $10^{th}$ symbol from the left end is a 0 and the $20^{th}$ symbol from the right end is a 1 is best described as

(a) regular set (b) CFL

(c) CSL (d) R.E. set

07. The minimal DFA that acts as a mod 3 counter and outputs a 1 whenever the input in binary treated as binary integer is divisible by 3 has

(a) 3 states (b) 4 states

(c) 5 states (d) none of the above

08. The minimal finite automata accepting the set of all strings over {0, 1} where the fourth symbol from the right end is a 1 has

(a) 4 states (b) 8 states

(c) 10 states (d) 16 states

09. The regular expression denoting the set of all strings not containing two consecutive 1's given by

(a) $(0+10)^* (\varepsilon +0)$ (b) $(0+10)^* (\varepsilon +1)$

(c) $(1+01)^*$ (d) $(\varepsilon + 0)(101)^* (\varepsilon+0)$

10. Let L = {Set of all binary strings whose integer equivalent is divisible by 4}. Then the number of states in Minimal FA accepting 'L' is

(a) 2 (b) 3

(c) 4 (d) 5

11. The regular expression with will strings 0' s and 1's with at lest two consecutive 0's is

(a) $(1+10)^*$ (b) $(0+1)^*00\ (0+1)$

(c) $(0+1)^*\ 011$ (d) $0^*1^*2^*$

12. Give the regular expression that derives all strings of a's and b's where each string contain even occurrences of substring ab.

(a) $(abab)^*$ (b) $(b + abab)^*$

(c) $(b + aa^*\ bb^*\ aa^*\ b)^*$ (d) $(b.aa^*bb^*aa^*b)^*$

13. Which of the following strings will the transition diagram given below recognize?

(a) Ending with 10 b) Not ending with 10

(c) Not ending with 01 d) None

keralanotes.com

14. The regular expression denoting the set of all binary strings not containing 000 as a substring is

(a) $(0+01+001)^*(\varepsilon + 0 + 00)$

(b) $(\varepsilon + 0 + 00)\ (1+10+100)^*$

(c) $(0+1)^*000(0+1)^*$

(d) none of the above

15. The minimum state automaton to recognize the set denoted by $(a+b)^*$ abb has

(a) 3 states b) 4 states

(c) 6 states (d) none of the above

16. What is the language expressed by the regular expression $(0^*011^*100^{*)*} \cup (1^*100^*011^{*)*}$

(a) The set of all strings, which contain equal number of 01's and 10's

(b) The set of all strings, which contain 01 or 10 as substring.

(c) The set of all strings that have same number of disjoint appearances of 10 and 01

(d) None of the above

17. Let $\sum$ = {a, b}. Which of the following regular expression define the language :

L = { w$\in\sum$*/w has atleast one substring ab}

(a) b*a*abb*a*

(b) (a$\cup$b)* (ab)*(b$\cup$a)*

(c) (a*b*)*ab(a*b*)*

(d) b*a*(ab$\cup\phi$*)b*a*

18. The minimal finite automata accepting the set denoted by (0+1)* (00+11) has

(a) 3 states b) 4 states

(c) 5 states d) 6 states

19. The regular expression

(aa)* + a(aa)* + aaaaa*a*

is the same as

keralanotes.com

(a) (a+aa+aaa)*

(b) aaa* + aaaaa* + aaaaaa*

(c) (aaa)*a****(a*+aa*)a+

(d) none of the above

20. The complement of the language

L = {$a^n$ $b^{n/}$ n ≠ 100} is _____

(a) Regular b) context free

(b) Context sensitive d) None

21. If P is a prime Number and a is positive integer such that $a^{p-1}$ = r (mod p), then r is

(a) a (b) 1/a

(c) 1 (d) $a^2$

22. The set of all strings over {0, 1} starting with 00 and ending in 11 is

(a) 0011 (b) 00(0+1)*+11

(c) (00)*(11)* (d) 0*1*

23. The language {$ww^R$ | w is in (a+b)*} is accepted by

a) DFA b) NFA

c) 2DFA d) multi tape, multithreaded turing machine

**Level – 2 Questions**

01. Choose the true statement :

(a) The minimal finite automata accepting the set of all strings over {0, 1} containing an even number of 0's and an even number of 1' s has 4 states.

(b) The minimal finite automata accepting the set of all strings not containing three consecutive 0's has 5 states.

keralanotes.com

(c) The minimal finite automata accepting the set of all strings in (0+1)* that start with a 0 or start with a 1 has 3 states.

(d) The minimal finite automata is not always unique.

02. Choose the true statement :

(a) The minimal finite automata accepting the set of all strings over {0, 1} where the 1000 symbol from the right end is a 0 has over a $10^{100}$ no. of states.

(b) The minimal finite automata accepting the set of all even length strings in (0+1+2+3)* has 4 states.

(c) The minimal finite automata accepting the set of all odd numbered strings over a finite alphabet which has between 100 to 200 symbols has 5 states.

(d) The minimal finite automata accepting the empty set is not unique.

03. The Mealy machine M which recognizes all strings over {a, b} where the last two symbols are the same

(a) no equivalent moore machine

(b) has a moore machine equivalent of 1000 states (minimal)

(c) such a Mealy machine does not exist

(d) has an equivalent Moore machine

04. Which of the following statements is false?

(a) the finite automaton in which the output is associated with state only is called a Moore machine.

(b) the finite automaton in which the output is associated with state and input is called a Mealy machine.

(c) the finite automaton in which the movement of read head is in either direction (left or right) is called a two way finite automaton

keralanotes.com

(d) None of the above

06. When an NFA is converted to an equivalent DFA, the number of states

(a) is the same

(b) is always more

(c) is always exponential to the number of states of the NFA

(d) is bounded by $2^n$, where n is the number of states of the DFA

06. Choose the true answer.

(a) Every formal language can be described using a one symbol alphabet.

(b) Every formal language can be described using a two symbol alphabet.

(c) Every formal language can be described without using $\varepsilon$.

(d) Every formal language does not have a finite description.

07. Consider the set of all strings over {0, 1} not containing two consecutive 0's. The regular expression for the same is

(a) (1+01)*(ε+0)

(b) (1+01)*(ε+0)(1+01)

(c) (1+01)*(ε+0) (1+01)*(ε+0)

(d) (1+01)*(ε+0) (1+01)*(ε+0)(1+01)*(ε+0)

08. The set of all strings over {0,1} where every block of 7 consecutive symbols contains at least 2 0's and 3 1's is

(a) a finite set.

(b) a regular set that is not finite

(c) a context free language that is not finite

keralanotes.com

(d) a.r.e. set that is not finite.

09. The minimal finite automata accepting the set of all strings over {0,1,2} that interpreted as a base 3 number is congruent to 0 modulo 7 and also congruent to 0 modulo 5 has

(a) 35 states

(b) 12 states

(c) 36 states

(d) none of the above

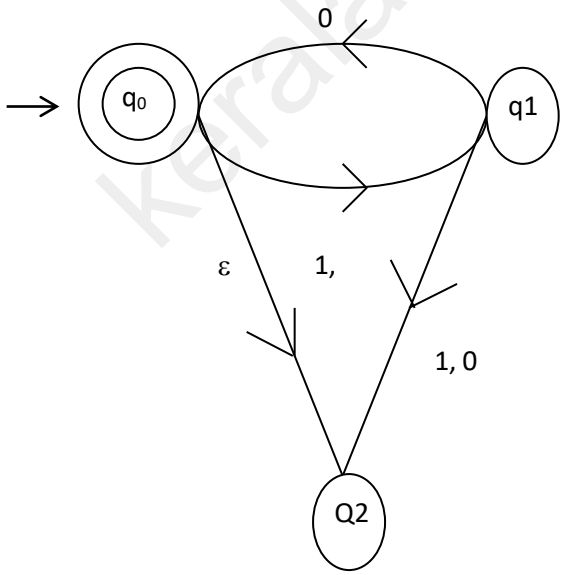
10. The minimal finite automata accepting the set of all strings over (0 + 1)* that do not have the sub string 0001 has

(a) 5 states b) 6 states

c) 7 states d) none of these

**Common Data for Q11 & Q12 is given below.**

Consider the transition Diagram.




11. Language accepted by FA is

(a) $L = \{(101)^n : n \geq 0\}$

(b) $L = \{(10)^n : n \geq 0\}$

keralanotes.com

(c) $L = \{10^n : n \geq 0\}$

(d) $L = \{(1^n0) : n \geq 0\}$

12. By referring to the above result, what happens when this automaton is presented with the string 111?

(a) It will be accepted

(b) It will be rejected by moving towards q1

(c) It will be rejected by moving towards dead state

(d) Both (b) & (c)

13. The minimal finite automata accepting the set of all strings over {0,1} starting with a 1 that interpreted as the binary representation of am integer are congruent to zero modulo 29 has

(a) 25 states (b) 26 states

(c) 29 states (d) 31 states

14. One of the following regular expressions is not the same as others. Which one?

(a) (a* + b*a*)*

(b) (a*b* + b*a*)* (a*b*)*

(c) ((ab)* + a *)*

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

15. The language {ww|w in (0+1)* and $|w|$<100000} is

A. a finite set B. regular set

C. CFL D. recursive set

(a) D>C>B>A (b) A>B>C>D

(c) B>D>C>A (d) C>B>A>D

keralanotes.com

16. Choose the non regular set

(a) $L = \{a^i b^j c^k \mid i < j < k < 10^{100}\}$

(b) $L = \{a^i b^j c^k \mid i < j < k < 10^{100}\}$

(c) $L = \{a^i b^j \mid j = I * I \text{ and } j < 10^{100}\}$

(d) $L = \{0^m 1^n 0^{m+n'} \mid m,n> = 1 \text{ and } m, n < 10^{100}\}$

17. Consider the languages $L_1 L_2$ and $L_3$ as given below.

$L_1 = \{0^p 1^q \mid p, q \in N\}$

$L_2 = \{0^p 1^q \mid p, q \in N \text{ and } p = q\}$ and

$L_3 = \{0^p 1^q 0^r \mid p,q,r \in N \text{ and } p=q=r\}$.

Which of the following statements is not

TRUE?

(a) Pushdown automata (PDA) can be used to recognize $L_1$ and $L_2$

(b) $L_1$ is a regular language

(c) All the three languages are context free

(d) Turing machines can be used to recognize all the languages

18. Consider the languages

**$L_1$ :** $\{www \mid |w| < 3000, w \in (a+b)^*\}$

**$L_2$ :** $\{a^n b^m \mid m > = 2000*n\}$

Choose the correct statement

(a) $L_1$ and $L_2$ are both regular

(b) $L_1$ is regular and $L_2$ is context free but nor regular

(c) $L_2$ is regular and $L_1$ is context free

(d) None of the above

keralanotes.com

19. Consider the regular expression identities

i. $r^* + s^* + t^* = (r + s + t)^*$

ii. $(rrss)^*rr = rr\ (ssrr)^*$

iii. $(r + s + t)^* = (r^*s^*t^* + \varepsilon)^*$

(a) all are valid

(b) (i) and (ii) are valid but (iii) is not

(c) (ii) and (iii) are valid but (i) is not

(d) (i) and (iii) are valid but (ii) is not

20. The minimal DFA accepting the set $(a+\varepsilon^*)\ (a+\varepsilon)$ has ____ states

(a) 1 (b) 2

(c) 3 (d) 4

21. The minimal finite automata accepting the set of all strings over {0, 1} where the number of 0's is divisible by three hundred and the number of 1's is divisible by two hundred has, choose the true statement

(a) 3000 states is the best case

(b) 40000 states or more

(c) 60000 states in the minimal machine

(d) 8000 states only in the minimal machine

22. Choose the correct statement

(a) The set of all formal languages is countable

(b) The set of all languages is not countable

(c) The regular sets are not countably infinite.

(d) The family of all finite automata is not countably infinite.

keralanotes.com

23. The transition function for the transition graph is

(a) $\varsigma : Q \times \Sigma^* \longrightarrow Q$

(b) $\varsigma : Q \times \Sigma^* \longrightarrow Q$

(c) $\varsigma : Q \times \Sigma^* \longrightarrow 2^{Q \times \Sigma}$

(d) All of the above

**Solutions**

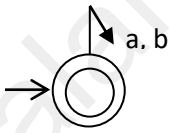
**Level – 1**

**01. Ans : (d)**

**Sol :** The operation of finite union & finite intersection preserve the regular sets. The set $\{\varepsilon\} \cup \{aa\} \cup \{aaa\} \ldots\ldots = a^*$ is the example of the intersection and union of regular sets being regular.

**02. Ans : (d)**

**Sol :** consider the NFA




**03. Ans : (a)**

**Sol :** All finite languages are regular.

**04. Ans : (a)**

**Sol :** The finite automata can be recursively enumerated.

**05. Ans : (b)**

**Sol :** It has 5 states

**06. Ans : (d)**

**Sol :** The set can be described by a regular expression.

keralanotes.com

**07. Ans : (a)**

**Sol :** A mod 3 counter needs 3 states in the minimal DFA.

**08. Ans : (d)**

**Sol :** This is the example of a DFA where the NFA to DFA requires an exponential number of states.

**09. Ans : (b)**

**Sol :** (a), (c) & (d) contain ε. Which is a negative string that does not contain two consecutive 1's.

**10. Ans : (b)**

**Sol :** The decimal 4 of L needs three bits to represent it. So a 3 state modulo machine is needed.

**11. Ans : (b)**

**Sol :** ……00…….. :2 0's are needed

So (0+1)* 00 (0+1)*

**12. Ans : (c)**

**Sol :** we need

[(a+b)*a (a+b)* b(a+b)* a(a+b)* b (a+b)*

**13. Ans : (b)**

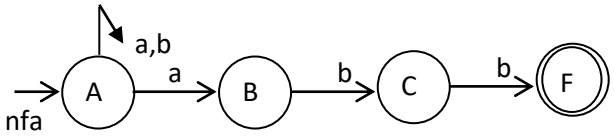
**Sol :** 10 is not recognized & 01 is recognized

**14. Ans (a) & (b)**

**Sol :** After ε, 0, or 00 after 1 appears. (c) contains 000.

**15. Ans (b)**

**Sol :**




keralanotes.com

**16. Ans (c)**

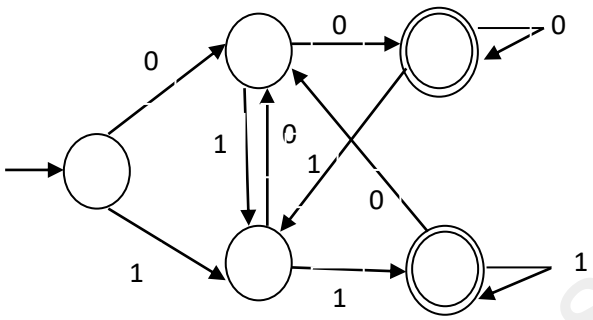
**Sol :** Use the method of elimination with the strings 0 & 1

**17. Ans : All of the choices**

**Sol :** The sub string ab is in all the choices (a) – (d)

**18. Ans (c)**

**Sol :**




**19. Ans : (a)**

**Sol :** (a) is a* which is the given R.E. (b) does not allow a & (c) does not allow a.

**Sol :** Use the method of elimination with the strings 0 & 1

**20. Ans : (c)**

**Sol :** The complement is a DCFL that is not regular. DCFLs are closed under complement.

**Sol :** Use the method of elimination with the strings 0 & 1

**21. Ans : (c)**

**Sol :** This is a statement of Fermat's Theorem

**22. Ans : (b)**

**Sol :** 00 _ _ start with 00

keralanotes.com

_ _ 11 end with 11

00 (0+1)* 11 start with 00 & end with 11

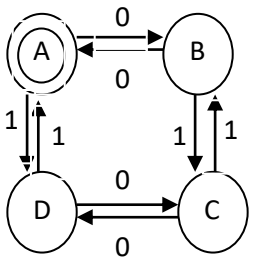
**23. Ans : (d)**

**Sol :** The language of palindrome is not regular or a DCFL. It is a CFL & hence R.E. So (d) is the answer.

**Level – 2**

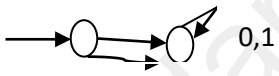
**01. Ans : (a)**

**Sol :** a)




(b) is false as a minimal FA is unique up to a homomophism.

(c) 0 (0+1)* + 1 (0+1)*




Is false

**02. Ans : (a)**

**Sol :** The DFA has $2^{1000}$ states & hence has more than a $10^{1000}$ no. of states.

**03. Ans : (d)**

**Sol :** Every moore machine has an equivalent mealy m/c & vice versa.

**04. Ans : (d)**

**Sol :** These are the standard definition of the Moore & Mealy machines & the two way finite automata

**05. Ans : (d)**

keralanotes.com

**Sol :** If NFA contains n states, then DFA contains atmost $2^n$ states.

**06. Ans : (c)**

**Sol :** $\varepsilon$ is just symbol so some other symbol like $\lambda$ can be used for the empty string.

**07. Ans : (a)**

**Sol :** After one or no 0's, a '1' compulsorily occurs.

**08. Ans : (b)**

**Sol :** Set = $(00)^+ + (111)^+ +$ other

So it is infinite. It can be expressed as a regular expression so it is regular.

**09. Ans : (d)**

**Sol :** The answer is a modulo 7 m/c multiplied by a modulo 5 m/c.

**10. Ans : (a)**

**Sol :** FA has 5 states

**11. Ans : (b)**

**Sol :** Evidently (1 0)* is accepted by the FA.

**12. Ans : (c)**

**Sol :** 111 will go to dead state

**13. Ans : (d)**

**Sol :** A modulo 29 machine has 31 states as it start with 1.

**14. Ans : (c)**

**Sol :** (a), (b) & (d) are (a+b)*

Where (c) does not contain 'b'

keralanotes.com

**15. Ans : (a)**

**Sol :** The given set is finite. Every finite set is regular. Every regular set is DCFL. Every DCFL is a CFL. Every CFL is DSL. Every CSL is recursive. Every recursive set is R.E.

**16. Ans : (b)**

**Sol :** By the Pumping Lemma for regular sets, (b) is not regular. The remaining choice are all finite sets.

**17. Ans : (d)**

**Sol :** $L_1$ requires no stack

$L_2$ requires one stack

$L_3$ requires 2 stack

So $L_1$ is Regular language, $L_2$ is CFL, and

$L_3$ is CSL and every RL, CFL, CSL are RES, accepted by TM

**18. Ans : (b)**

**Sol :** $L_1$ is a finite set & hence is regular. $L_2$ is a CFL that is not regular.

**19. Ans : (c)**

**Sol :** (ra)* r = r(ar)*

So (ii) is true

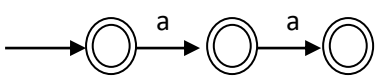
$(r + s + t)^* = (r^*s^*t^* + \varepsilon)^*$

$= (r^*s^*t^*)^*$

$= (r + s + t)^*$

**20. Ans : 3**

**Sol :** The minimal DFA is




keralanotes.com

**21. Ans : (c)**

**Sol :** $M_1$ 0' need 300 states

$M_2$ 1' need 200 states

$M_3 = M_1 \times M_2 = 60000$ states

**22. Ans : (b)**

**Sol :** Set of all formal languages is uncountable, since there is no one – to – one relation between set of formal languages and N.

**23. Ans : (c)**

**Sol :** The Question deals with the conversion of a transition diagram having ε-moves. The non-determinism requires all possibilities to be considered.

**Chapter – 2**

**Context – free Languages & Push down Automata**

**Level – 1 Questions**

01. The intersection of two CFL's

(a) is always a CFL (b) may be a CFL

(c) is never a CFL (d) is always a CSL

02. Which of the following statements is true?

(a) if a language is context free it can always be accepted by a deterministic push down automaton

(b) The union of two CFLs is a CFL

(c) The context free languages are closed under intersection

(d) The complement of a CFL is always a CSL

03. Consider the following two languages

keralanotes.com

$L_1 = \{1^n 0^n 1^n 0 / n \geq 0\}$

$L_2 = \{a^n b^k / n \leq k \leq 2n\}$

Which of the following statement is true.

(a) both $L_1$ and $L_2$ are context free

(b) $L_1$ is context free but not $L_2$

(c) $L_2$ is context free but not $L_1$

(d) neither $L_1$ nor $L_2$ is context free

04. The language generated by the following context free grammar is,

$S \rightarrow AB$, $A \rightarrow a\,A/a$, $B \rightarrow bBc/\varepsilon$

(a) $\{a^m b^n c^n / m \geq 0, n \geq 0\}$

(b) $\{a^n b^m c^n / m \geq 0, n \geq 0\}$

(c) $\{a^n b^n c^m / m \geq 0, n \geq 0\}$

(d) None of the above

05. The recursive sets, $CFL_s$ and R.E. sets are closed under

(a) Union and complement

(b) Intersection and complement

(c) Union and intersection

(d) Union

06. CFL is not closed under

(a) Intersection

(b) complement

(c) difference

(d) all the above

07. Which of the following language cannot be accepted by any deterministic push down automaton

keralanotes.com

(a) The set of all strings over {a, b} consisting of equal number of a's and b's.

(b) A language of palindromes

{x∈{a, b}* / x = rev(x)}

(c) {wcw$^R$/ w∈{a, b}*}

(d) all of the above

08. The language which is accepted by LBA is called as

(a) Regular

(b) Context free

(c) Context dependant

(d) None of the above

09. Choose the correct statement

(a) The nondeterministic & deterministic finite automata are not equivalent.

(b) The nondeterministic & deterministic PDA are not equivalent.

(c) The nondeterministic & deterministic TMs are not equivalent.

(d) The nondeterministic & deterministic halting TMs are not equivalent.

10. In the case of non – determinism choose the incorrect statement

(a) Deterministic and nondeterministic finite automata are equivalent in power.

(b) Deterministic and nondeterministic push down automata are equivalent in power.

(c) Deterministic and non deterministic two push down tape machines are equivalent in power.

(d) Deterministic and nondeterministic three counter machines are equivalent in power.

11. The language $0^* \cup \{0^n1^n/n\geq1\} \cup \{0^n1^{2n}/n\geq1\}$ is accepted by

(a) 2 DFA

(b) DPDA

(c) PDA

(d) LBA but not by a PDA

keralanotes.com

12. Consider a PDA M accepting a CFL L. Choose the correct statement.

    (a) A PDA can never loop on its input and makes as many moves as the size of the input.

    (b) In a PDA the maximum size of the stack for an input of size n, is a polynomial p (n).

    (c) In a PDA the maximum size of the stack for an input of size n, is an exponential of $O(2^n)$

    (d) A PDA can be defined in two equivalent models having acceptance by final state or acceptance by empty store.

13. Consider the CFG G generating the CFL $L = \{ww^R \mid w \in \{a, b\}^*\}$. Choose the correct statement.

    (a) As L is a CFL, g must be ambiguous.

    (b) L is CFL that is also a DCFL.

    (c) If L is defined over a single alphabet {a} then L is accepted by non deterministic finite automata.

    (d) the exists an LR (k) that generates L.

14. Let L be a language over the alphabet {a} & be generated by some CFG G. Choose the correct statement.

    (a) L is a CSL that is not a CFL

    (b) L is a CFL that is not a DCFL

    (c) L is a DCFL that is not regular

    (d) L can be denoted by some regular expression

15. Consider the language $L = \{ww \mid w \in \sum^*\}$.

    (a) L is a CSL that is not a CFL as down by the pumping Lemma for CFLs.

    (b) L is a CSL as shown by the pumping Lemma for CFLs.

    (c) L does not model the definition before use requirements in some programming languages.

    (d) L can be described by extended regular expression which arranges the standard definition of regular expression with the intersection & complement operators.

16. Consider the language

    $L = \{a^i b^j \; a^i b^j \mid i,j \geq 1\}$.

keralanotes.com

Choose the correct statement.

(a) L can be accepted by some PDA

(b) L can be described by the intersection of regular expression.

(c) L models the requirement that the number of parameters in procedure definition & use be the same.

(d) The pumping Lemma for regular sets shows that L is a CSL.

17. Consider the languages

i. $L_1 = \{a^n b^{2n} | n \geq 1\}$

ii. $L_2 = \{a^n b^{2n} | n \geq 1\}$

Choose the false statement

(a) $L_1$ is a DCFL

(b) $L_2$ is a DCFL

(c) $L_1 \cup L_2$ is accepted by some PDA

(d) $L_1 \cup L_2$ is a DCFL

18. Consider the languages

$L_1 = \{a^n b^n c^i | n, I \geq 1\}$ &

$L_2 = \{a^m b^j c^j | m, I \geq 1\}$ &

Choose the false statement.

(a) $L_1$ & $L_2$ are CFLs

(b) $L_1$ may have an ambiguous CSG generating it

(c) $L_2$ may have and ambiguous CFG generating it.

(d) $L_1 \cup L_2$ is not inherently ambiguous.

19. Define a linear grammar G having rules of the form

$A \longrightarrow a$, $A \longrightarrow aB$

$A \longrightarrow Ba$, $A \longrightarrow B$,

$A \longrightarrow ab$.

Where A, B are non-terminals. & a, b are terminals, Let G generates a CFL L.

Choose the correct statement

(a) Every CFL does not have a linear grammar.

keralanotes.com

(b) Every regular set does not have a linear grammar.

(c) A linear grammar does not describe all the languages that can be denoted by regular expressions.

(d) A linear grammar can generate the set $\{a^n b c^n \mid n \geq 1\}$

**Level – 2 Questions**

01. The language $\{ww^R\ w\ w\ w\ w\ w\ w^R\ w\ w\ w\ w\ w^R \mid w \text{ in } (a+b)^*\}$ is accepted by

(a) PDA

(b) NFA

(C) 2DFA

(d) multitape, multiheaded turing machine

02. **Statement (I) :** The language L = set of all strings not containing 101 as a substring is regular set.

**Statement (II) :** L satisfies the pumping Lemma for regular sets.

(a) I is true, II is false
(b) I is false II is true
(c) I & II are true
(d) I & II are false

03. A is given a CFG, $G_1$ that generates language F is not a DCFL $L_1$ & a LR(k) grammar $G_2$ that generates a languages $L_2\$$. Choose the correct statement

(a) For every $G_1$ there exists a grammar $G_2$.

(b) Every $G_1$ & $G_2$ are unambiguous.

(c) $L_1$ $L_2$ is always a DCFL.

(d) $L_2$ is always a DCFL if \$ is a symbol not in the vocabulary

04. A DPDA's that is accepted by empty store. Choose the correct statement.

keralanotes.com

(a) For every DCFL, there exists a DPDA that accepted by empty store.

(b) For every DCFL that satisfies the prefix property, there does not exist, a DPDA that is accepted by empty store.

(c) If L is a DCFL then L$ can be generated by some LR(k) grammar.

(d) Every DCFL can be defined to be accepted by an empty store or final sate.

05. Consider the machines

$M_1$ = a NPDA accepting $L_1$

$M_2$ = a 2 way PDA accepting $L_2$

$M_3$ = a PDA with two stacks accepting $L_3$.

Choose the false statement

(a) For any CFL L, there exists some $M_1$, $M_2$, & $M_3$.
(b) For some CSL L, there exits some $M_1$, $M_2$, & $M_3$.
(c) For every recursive set, there exits some $M_2$, & $M_3$.
(d) The power of the machines are not $M_1 \leq M_2 < M_3$.

06. Consider the properties of a context – free grammar G in the chomsky normal from (CNF), the Greibach Normal From (GNF) & an operator grammar (OG). Choose the false statement.

(a) In CNF, the r.h.s of a rule is either a terminal or two adjacent non – terminals.

(b) In GNF, the r.h.s of a rule is either a terminal or a terminal followed by a string of non – terminals

(c) In OG, the r.h.s of every rule does not have two adjacent non – terminals.

(d) No grammar cannot be in CNF, GNF & OG at the same time

keralanotes.com

07. We are given a type 0 grammar $G_0$, a type 1 grammar $G_0$, a type 1 grammar $G_1$, a type 2 grammar $G_2$, & a type 3 grammar $G_3$, generating the formal language $L_0$, $L_1$, $L_2$ & $L_3$a respectively.

Choose the true statement,

(a) $G_0$ cannot generate all possible $L_1$

(b) $G_1$ cannot generate all possible $L_2$

(c) $G_2$ cannot generate all possible $L_3$

(d) $G_3$ cannot generate all the finite sets

08. We are given a PDA $M_1$ accepting $L_1$ & a PDA $M_2$ accepting $L_2$.

Choose the false statement.

(a) We can construct a PDA $M_3$ that accepts all the strings $M_1$ & $M_2$ accept.

(b) We cannot construct a PDA that accepts some strings accepted by both $M_1$ & $M_2$ .

(c) We cannot construct a PDA that accepts string $\overline{L_1}$, $\overline{L_2}$.

(d) We can construct a PDA that accepts all the strings not in $L_1$.

09. Consider the languages

i. $L_1 = \{a^n b^{2n} \mid n \geq 1\}$

ii. $L_2 = \{ca^n b^n \mid n \geq 1\}$

Choose the false statement.

(a) $L_1$ is accepted by some DPDA

(b) $L_2$ is accepted by some DPDA

(c) $L_1 \cup L_2$ is not accepted by any DPDA

(d) $L_1 \cup L_2$ is not accepted by some PDA

keralanotes.com

10. Consider the languages

i. $L_1 = \{ww^R \mid w \in \sum^*\}$

ii. $L_2 = \{w \not\subset w^R | w \in \sum^*\}$

iii. $L_3 = \{wxw^R | w,x \in \sum^*\}$

When $\sum = \{a, b\}$.

Choose the false statement.

(a) $L_1$ cannot be accepted by any DPDA & needs a PDA or LBA to accept it.

(b) $L_2$ can be accepted by a DPDA but not a finite automata.

(c) $L_3$ is a sample regular language {a, b}+ {a, b} {a, b}+ & can be accepted by some finite automata.

(d) All $L_1$, $L_2$ & $L_3$ are inherently ambiguous languages i.e. any CFG constructed for them will be necessarily ambiguous.

11. We are given the regular sets $R_1 = \varepsilon^*$ & $R_2$ = set of all strings over {a, b} where the $11^{th}$ symbol form the right end is a. We are also given an arbitrary CFL $L_1$ & an arbitrary DCFL $L_2$.

Choose the false statements.

(a) $L_1 = R_1$ or $L_1 = R_2$ are both undecidable

(b) $L_2 = R_1$ or $L_2 = R_2$ are both decidable

(c) $L_1 \cap L_2 \cap R_1$ has an emptiness problem that is undecidable.

(d) $R_1 = R_2$ is undecidable

12. Consider the following machine

(i) A PDA $M_1$ where the stack size cannot be more than a function f of the input size n.

(ii) A DPDA $M_2$ where the stack size bounded by some prime number.

keralanotes.com

(iii) A ODA $M_3$ where the maximum stack size should be m! for some integer m.

Choose the false statement

(a) $M_1$ accepts only regular sets

(b) $M_2$ accepts only regular sets

(c) $M_3$ can accepts $CFL_s$

(d) None of the above

13. Consider the language $L_1 = \{a^n b^{2n} \mid n \geq 1\}$, the homomorphism h([p) = a, h (q) = aa, Let $L_2 = h^{-1}(L_1)$ & R = p* q*.

Choose the false statement

(a) $L_1 \cap R = \{a^m b^m \mid m \geq 1\}$

(b) $L_2$ & $L_1$ are both CFLs that are not regular.

(c) $L_2$ is not a CFL but is a CSL.

(d) None of the above.

**Solutions**

**Level -1**

**01. Ans : (b) & (d)**

**Sol :** 1. $a^* \cap b^* = \phi$ a CFL

2. The CFLs are not closed under intersection.

**02. Ans : (b) & (d)**

**Sol :** The DCFLs are a smaller class than the CFLs. The CFLs are closed under union but not under intersection. Every CFL is a CSL. Complement of CFL is recursive

**03. Ans : (a)**

**Sol :** (b) $L_1$ is a standard CSL

keralanotes.com

(c) $L_2$ is CFL

**04. Ans : (d)**

**Sol :** $S \longrightarrow AB$

$A \xrightarrow{*} a^+s$

$B \xrightarrow{*} \{b^n c^n / n \geq 0\}$

Note that $\varepsilon$ is not generated by the grammar

**05. Ans : (d)**

**Sol :** The CFLs are not closed under intersection or complement.

**06. Ans : (d)**

**Sol :** A CFL is not closed under intersection, or complement & hence difference

**07. Ans : (b)**

**Sol :** The language L= $\{ww^R / w \in \Sigma^*\}$ is a CFL but not a DCFL.

**08. Ans : (c)**

**09. Ans : (b)**

**Sol :** The PDA are not the same for non determinism.

**10. Ans : (b)**

**Sol :** The DPDA & PDA are not equivalent. Non-determinism does not add any power to the finite automata or the TM.

**11. Ans : (c)**

**Sol :** The language is a CFL but not a DCFL.

**12. Ans : (d)**

**Sol :** Any CFL can be accepted by a PDA that can be accepted by final state. This has an equivalent model of acceptance by final state.

keralanotes.com

For (a), (b), (c) we note that a PDA can make an infinite number of moves on ε input, & for each one of these moves, the store can grown in size. So in general the PDA needs an infinite push down store.

**13. Ans : (c)**

**Sol :** (c) if the vocabulary is {a} then L = (aa)* which is a regular set

(b) L is a standard CFL, the language of all palindromes is not a DCFL.

(d) G is unambiguous. G is a partially CFG. Ambiguity problem is for all CFGs.

(d) A L is not a DCFL, no LR (k) can generate it.

**14. Ans : (d)**

**Sol :** CFLs over a single alphabet are regular.

**15. Ans : (a)**

**Sol :** L is a standard CSL that is not a CFL (& hence not regular). This can be shown using the pumping Lemma for CFLs. Also it models the definition before use requirement in programming languages.

**16. Ans : (c)**

**Sol :** L is a CSL that is not a CFL. It cannot be shown to be a CFL by using pumping Lemma for CFLs. A LBA can be constructed by accepting L & so it is a CSL. L models the requirement of the same procedure to have the same number of parameters.

**17. Ans : (d)**

**Sol :** We can construct DPDA's to accept $L_1$ & $L_2$. We can construct a nondeterministic PDA to accept $L_1 \cup L_2$. However $L_1 \cup L_2$ will require non-determinism & cannot be a DCFL.

**18. Ans : (d)**

**Sol:** It can be shown that $L_1$ & $L_2$ are CFLs by constructing CFG's generating them. Ambiguity can be artificially added to them to make them generated by ambiguous CSG's.

Keralanotes.com

It runs out that $L_1 \cup L_2$ is inherently ambiguous. As strings $\{a^n b^n c^n\}$ will have two derivations.

**19. Ans : (a)**

**Sol:** The languages generated by linear grammars are a proper subset of the CFLs. The linear grammar S aSc/b generates $\{a^n b\, c^n \mid n \geq 1\}$

All right linear grammars & all left linear grammars are trivially linear grammar. So all regular sets can be generated by linear grammars. So all regular expressions can be denoted by linear grammar.

**Level – 1**

**01. Ans : (d)**

**Sol:** The language is a CSL & hence a R.E. set accepted by a TM or its Vice versa.

**02. Ans : (b)**

**Sol:** satisfying the pumping Lemma give no conclusion.

$\therefore$ L is a regular set that satisfies the Pumping Lemma.

**03. Ans : (d)**

**Sol :** (a) not true as $G_1$ every CFL is not a DCFL. So every CFG $L_1$ does not have an equivalent LR(k) grammar $G_1$.

(b) LR(k) grammars are unambiguous, but $G_1$ being an arbitrary CFG it may be ambiguous.

(c) The concatenation of two CFLs is a CFL that need not be a DCFL.

(d) If L is a DCFL the L$ is LR(k) for some k.

**04. Ans : (c)**

**Sol :** The DPDA that is accepted by final state accepts DCFLs. If the prefix property is satisfied we can use DPDA's that is accepted by empty store.

keralanotes.com

(c) Standard theorem. If L is a DCFL then L$ can be generated by some LR (k) grammar.

**05. Ans : (d)**

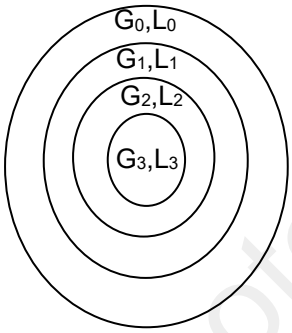
**Sol.** $M_1$ accepts all CFLs. $M_2$ accepts some CSLs. $M_3$ is a variation of the Turing machines.

**06. Ans : (d)**

**Sol :** Consider the grammar $S \rightarrow a$. It is in CNF, GNF & OG at the same time.

**07. Ans : (d)**

**Sol :**




**08. Ans : (d)**

**Sol :** (a) The union of two CFLs is a CFL.

(b) The intersection of two CFLs need not be a CFL.

(c) The complement of a CFL need not be a CFL & the CSLs are closed under concatenation.

(d) The CFLs are not closed under complement.

**09. Ans : (c)**

**Sol:** $L_1$ is a DCFL as a DPDA can be constructed to accept it. $L_2$ also can be accepted by an easily constructed DPDA. Since $L_1$ & $L_2$ are CFl$_s$. $L_1 \cup L_2$ is a CFL & can therefore be accepted by some PDA.

$L_1 \cup L_2$ is a DCFL as DPDA can use the presence of the first symbol 'c' to table the non determinism.

keralanotes.com

**10. Ans : (d)**

**Sol :** As (c) is a regular set it will have unambiguous left linear or right linear grammar generating it.

As $L_2$ is a DCFL it can be accepted by DPDA. $L_1$ is a standard CFL that is not a CSL.

**11. Ans : (d)**

**Sol :** It is decidable a DCFL $L_2$ is equivalent to a given regular set R. This is undecidable for DCFLs.

The intersection of two DCFLs can given all the valid computations of a turing machine & so is the same as the halting problem.

It is decidable if two regular sets are equivalent.

**12. Ans (c)**

**Sol :** For $M_1 \cup M_2$ the stack size is bounded. So the stack can be stored in the finite control. So $M_1 \cup M_2$ accept only regular sets.

For $M_3$ the stack is not bounded. At the most we have to add the stack with dummy symbols so that the stack is m! for some m. So this is the general definition of a PDA and accept all the CFLs.

So for $M_3$ also the stack size is bounded by m!. Hence the memory is finite and can be stored in finite. Hence $M_2$ accepts only regular set.

**13. Ans : (c)**

**Sol:** L is a CFL. The CFLs are closed under homomorphism & inverse homomorphism, so $h^{-1}(L_1) = L_2$ is a CFL. $L_2$ has p's & q's jumbled up intersection with R arrange the p's before the q's. As a's, (bb)'s are the same in $L_1$ they are the same number of p's and q's in any string in $L_2$.

keralanotes.com

## Chapter – 3

## Turing Machines, Modifications, CSLs, Recursive & R.E. sets

### Level – 1 Questions

01. To evaluate expressions in C without function calls, which of the following is mandatory?

(a) one stack is enough

(b) two stacks are must

(c) unlimited number of stacks are required

(d) a turing machine is required and it must be non deterministic in the general case

02. Some how I proved that P = NP then which of the following statements is true?

(a) NP is closed under complement

(b) NP – CoNP (complement of NP)

(c) Both a and b are true

(d) None of the above is true

03. Recursive sets are closed under

(a) Kleene closure

(b) Substitution

(c) Homomorphism

(d) Inverse homomorphism

04. Consider the following languages

$L_1$ = {<M> : L(M) contains a word of length < 1000}

keralanotes.com

$L_2$ = {<M> : M) accepts some word within 1000 steps of computation}

(R⟶ Recursive, RE⟶ recursively enumerable)

Which of the following statements is true?

(a) $L_1 \in R$ and $L_2 \in RE$
(b) $L_1 \in R$ and $L_2 \in RE$
(c) $L_1 \in RE$ and $L_2 \notin RE$
(d) $L_1 \in RE$ and $L_2 \in RE$

05. The language generated by the grammar S ⟶aSa|bSb|d is accepted by a

A. DPDA B. PDA

C. LBA D. Turing machine

(a) D>C>B>A (b) A>B>C>D

(c) B>C>D>A (d) C>D>B>A

06. The language consisting of an equal number of a's and b's can be accepted by a

A. DPDA B. PDA

C. turing machine D. LBA

(a) C > D > B > A (b) A > B > C > D

(c) B > C > D > A (d) D > C > B > A

07. If a 2DFA we supply an unlimited amount of ink so that it can write on its input tape, it become a

(a) DPDA (b) PDA

(c) LBA (d) Turing machine

08. A finite set that is accepted by a finite automata is always

(a) type 0 but not type 1

keralanotes.com

(b) type 1 but not type 2

(c) type 2 but not type 3

(d) always type 3

09. Which of the following languages are closed under complement?

(a) CFL

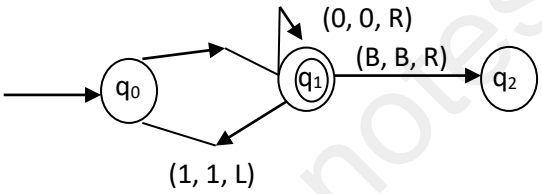
(b) CSL

(c) Recursive sets

(d) Recursively enumerated sets

**Statement for linked answer Q 10 to Q 11 is given below :**

Let M be the Turing Machine given below :




10. The languages accepted by M is

(a) $\{00^n | n \geq 0\}$

(b) $\{0^n10^n1 \mid n \geq 1\}$

(c) $\{0^n1\ 0^n1 \mid n \geq 1\}$

(d) $\{0^n10^n1 \mid n \geq 1\}$

11. The above language is

(a) Regular language (b) DCFL

(c) CFL (d) CSL

12. Choose the correct statement

(a) The recursive sets can be effectively enumerated

keralanotes.com

(b) The formal language can be effectively enumerated.

(c) All real numbers can be effectively enumerated

(d) The finite automata can be effectively enumerated

13. Choose the correct statement

(a) If any problem in P is closed under complement, then P = NP.

(b) If every problem in NP is closed under complement, then P = NP.

(c) If every problem in NP is NP – complete, then P = NP.

(d) If any problem in NP is not NP – complete, then P – NP.

14. Choose the undecidable problem.

(a) Equivalence of finite automata

(b) Equivalence of regular expression

(c) Emptiness problem of CSLs

(d) Emptiness problem of R.E sets

15. The following are not countable

(a) the subsets of all regular sets

(b) the subsets of all CFLs

(c) The subsets of all CSLs

(d) All of the above

16. The following problems are not decidable

(a) Whether a CFL is infinite or finite

(b) The membership problem of CSL's.

(c) The membership problem of recursive sets

(d) Whether a CFL consists of all strings over the terminal vocabulary.

keralanotes.com

17. The union of the classes of regular sets and context sensitive languages is

(a) not countably infinite

(b) countably infinite

(c) sometimes countably infinite

(d) none of the above

18. Let $M_1$ be a deterministic finite automata accepting a language $L_1$ & let $M_2$ be a non – deterministic finite automata accepting a language $L_2$.

Choose the correct statement :

(a) $L_1$ is equal to $L_2$

(b) It is decidable if $L_1 = L_2$

(c) There is no algorithm to decide if $L_1$ is empty

(d) The intersection of $L_1$ & $L_2$ is always a finite set

19. Let $M_1$ be a deterministic push down automata accepting a DCFL $L_1$. Let $G_2$ be an LR(k) grammar generating the language $L_2$. Let $M_2$ be a two way deterministic finite automata (2DFA) accepting the language $L_3$.

Choose the correct statement.

(a) There is no algorithm to decide if $L_1 \cap L_2$ is empty.

(b) $L_1 = L_2$

(c) $L_2 = L_3$

(d) The problem of whether $L_1 = L_2$ is undecidable

20. Let M1 be a push down automata accepting a language $L_1$.

Choose the correct statement :

(a) $L_1 = \phi$ is decidable

keralanotes.com

(b) $L_1 = \sum^*$ is decidable

(c) $L_1$ is a finite set is not decidable

(d) $L_1$ is an infinite set is not decidable

21. Given an arbitrary C program $C_1$ and a JAVA program, $J_2$. Choose the correct statement.

(a) $C_1$ may not be an algorithm but $J_2$ is always an algorithm.

(b) $C_1$ may not be an algorithm but $J_2$ may not be an algorithm.

(c) Both $C_1$ & $J_2$ are algorithms always.

(d) neither $C_1$ nor $J_2$ necessarily an algorithm.

22. Consider an arbitrary C program which has some assignment statement a. Choose the correct statement.

(a) It is decidable whether a is executed.

(b) It is undecidable whether a is live or dead always

(c) It is decidable if a is repeatedly executed in some iteration.

(d) None of the above

23. Choose the correct statement

(a) The graph coloring problem is undecidable.

(b) The register optimization problem in compilers can be reduced to graph coloring problem is undecidable.

(c) The graph coloring problem with less than 10 nodes is intractable.

(d) The graph coloring problem is decidable and seemingly intractable.

24. A is given a recursive set L accepted by some LBA M & a homomorphism h. Choose the correct statement.

keralanotes.com

(a) It may not be possible to decide if L = $\phi$ but it is always possible to decide if h(L) = $\phi$

(b) It is undecidable if h (L) is a R.E. set

(c) The emptiness problem of h(L) is the same as the halting problem of turing machine

27. A considered the sets of Turing machines describing all the Hamiltornian cycle problems. Choose the correct statement.

(a) The membership problem of S is decidable

(b) It is decidable if the complement of S is empty

(c) It is decidable if S is empty, finite or infinite

(d) It is decidable if S = $\sum^*$

**Level - 2 Questions**

01. A single tape turning machine M has two states $q_0$ and $q_1$, of which $q_0$ is the starting state. The tape alphabet of M is {0, 1, B} and its input alphabet is {0, 1}. The symbol B is the blank symbol used to indicate end of an input string. The transition function of M is described in the following table.

| | 0 | 1 | B |
|---|---|---|---|
| $q_0$ | $q_0$, 1, R | $q_0$, 1, R | $Q_1$, B, R |
| $q_1$ | $q_0$, 1, R | $q_0$, 0, R | Halt |

Which of the following statements is true about M?

(a) M accepts all strings that end with 0 only

(b) M accepts all strings that end with 1 only

(c) M accepts all strings in $(0+1)^+$ only

(d) M accepts all strings in $(0+1)^*$ 01 $(0+1)^*$ + $1^*0^*$

keralanotes.com

02. Match the following

| Grammar | Classification |
|---|---|
| P.S→ aS\|bS\|a,<br>S→ A,<br>A→ a | 1.Type 3 and right linear |
| Q.S→aAb\|ab,<br>C→ c/cC<br>S→ AC/abc | 2. Type 2 and not generating a regular set |
| R.S→ LWaR<br>Wa →Waaaa<br>WR→W1 R<br>aW1→ W1a<br>LW1→ LW<br>WR →W2R<br>aW2R→ W2Raaaa<br>LW2R→ ∈ | 3. Type 0 and not generating a CFL |
| S.S → aSBC\|aBC<br>CB →BC<br>aB → ab<br>bB → bb<br>bC → bc<br>cC → cc | 4. Type 1 and not generating a CFL |
| | 5. LR (k) |
| | 6. Type 3 and left linear |

**Codes :**

(a) P-1. Q=5. R-3, S-4

(b) P=6, Q-4, R-3, S-5

(c) P-1, Q-3, R-6, S-2

(d) P-1, Q-2, R-3, S-6

03. Choose the true statement.

(a) The regular sets, R.E. sets, CSLs, DCFLs and CFLs are closed under intersection

(b) The regular sets, recursive sets, DCFLs and CFLs are preserved under the operation of complement

(c) The regular sets are closed under intersection but the CFLs are not

(d) The CFLs are closed under complement but the regular sets and R.E. sets are not

keralanotes.com

**Statement for Linked answer Q.04 and Q.05 is given below.**

Ram obtains a turing machine that can accept the empty set $L_n$. Shyam obtains a turing machine that can accept the set of all strings over the input alphabet $L_m$. Thomas studies $L_n$ to classify it in the Chomsky hierarchy. Rahim construct a turing machine to accept $L_1$ the complement of $L_n$.

04. Choose the correct statement :

(a) Thomas will not be able to determine if $L_n$ is a regular set.

(b) Thomas will not be able to determine all strings in $L_n$ as $L_n$ is a recursive set.

(c) Shyam will not be able to determine if $L_m$ is the same as $L_n$ as the equivalence of turing machines is undecidable.

(d) Shyam will not be able to determine if $L_m$ is a subset of $L_n$ as the equivalence problem of turing machines and the containment problem is undecidable.

05. Choose the correct statement :

(a) Rahim will not be able to determine $L_1$ as $L_1$ is not R.E.

(b) Rahim will not be able to determine $L_1$ as $L_1$ is recursive.

(c) Rahim will not be able to determine $L_1$ as $L_1$ is R.E.

(d) Rahim will not be able to determine $L_1$ as $L_n$ is a finite.

06. Consider the complexity classes P, NP, PSPACE, EXPTIME, then which of the following relationship is true?

(a) $P \subset NP \subset PSPACE \subset EXPTIME$

(b) $P \subset NP \subset EXPTIME \subset PSPCE$

(c) $P \subset PSPACE \subset NP \subset EXPTIME$

(d) $NP \subset EXPTIME \subset P \subset PSPACE$

keralanotes.com

07. Which of the following statement is false

(a) A Turing machine is more powerful than a finite state machine because it has halt state.

(b) A finite state machine can be assumed to be a Turing machine of finite tape length, with rewinding capability and unidirectional tape movement.

(c) both (a) and (b)

(d) none of the above

08. Choose the false statement

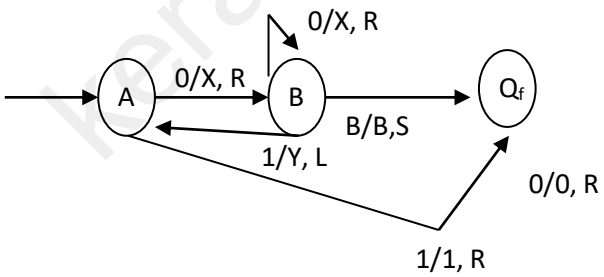
(a) Every CSL can be accepted by a finite automata if we allow an unbounded number of states.

(b) The complement of the language L = {ww|w$\in$ {a, b}$^*$} is a CFL.

(c) The complement of the language L = ($a^n$ $b^n$ $c^n$| n $\geq$ 1} is a CFL

(d) Every regular expression does not denote a regular set

**Common Data for Q09 and Q10 is given below.**




09. The above TM accepts

(a) $0^*$ b) $1^*$

(c) $(0+1)^+$ (d) $(0+1)^*$

10. The language accepted by the above TM is

(a) finite

(b) recursive set

(c) CSL but not CFL

(d) CFL but not regular

**Common Data for Q11 & Q12 is given below.**

Consider a language $L_1$. $L_1 = \{0, 1)^*$ if another language $L_2$ is a CFL.

$L_1$ = empty set if another language $L_2$ is not a CFL.

$L_2$ = a language that is R.E. and its complement is R.E.

11. $L_1$ is

(a) regular and not finite

(b) CFL and not regular

(c) CSL and not a CFL

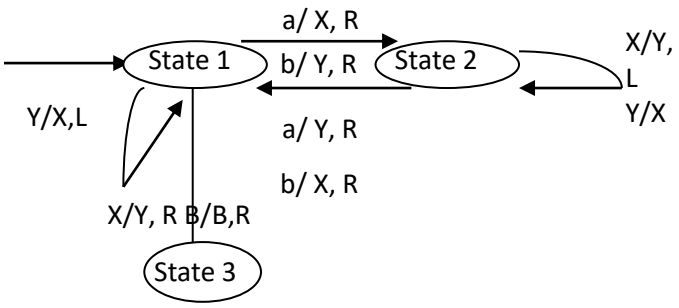
(d) r.e and not recursive

12. $L_2$ is

(a) empty (b) regular

(c) CFL (d) r.e

13. Consider the following Turing machine M :




Assume that initially tape contains the strings over the alphabet {a, b} and can end of the string is identified with the help of continuous blank symbols after the given string.

keralanotes.com

The above Turing machine M accepts

(a) All strings of a's and b's including ε.

(b) All strings of a's and b's including ε.

(c) All odd length strings over {a, b}.

(d) All even length strings over {a, b}.

14. Choose the correct statement in relation to a given instance P of a post corresponding problem.

(a) P can be simulated by the intersection of two deterministic push down automata.

(b) P can be simulated by the intersection of two regular sets.

(c) P cannot be reduced to the emptiness problem of context sensitive languages (CSLs)

(d) P can be simulated by the intersection of two languages $L_1$ & $L_2$, where $L_1$ is generated by an LR(k) grammar & $L_2$ is a language for which a description as a regular expression R exists.

15. Consider a language $L_1$ generated by a CSG $G_1$ & a language $L_2$ accepted by a non-deterministic LBA $M_2$. Choose the correct statement.

(a) It is undecidable whether $L_1$ is empty, finite, infinite of the same as $\Sigma^*$.

(b) It is undecidable if a given strings w is generated by $G_1$ or accepted by $G_1$ or accepted by $M_2$.

(c) It is decidable if $L_1 = L_2$ or $L_1 \leq L_2$.

(d) It is decicable if $L_1$ or $L_2$ is accepted by some two way nondeterministic finite automata (2NFA)$M_3$.

16. Consider a language $L_1$ accepted by deterministic Turing Machine $M_1$ & a language $L_2$ accepted by a two push down tape machine $M_2$. Chose the correct statement.

keralanotes.com

(a) The membership problem of $L_2$ & $L_1 \cap L_2$ is decidable.

(b) The equivalence problem of $L_1$ & $L_2$, viz. $L_1$ viz. $L_1 = L_2$ can be resolved by some algorithm.

(c) It is possible that $L_1 = L_2 = \sum^*$.

(d) It is decidable if $L_1$ & $L_1$ are regular sets.

17. Consider a CFG C1 generating syntactically correct C programs & a context free grammar $J_1$ generating syntactically correct JAVA programs.

Choose the correct statement.

(a) It is decidable if $C_1$ & $J_1$ decidable all and only algorithms

(b) It is undecidable if $C_1$ is ambiguous

(c) It is decidable if $J_1$ is ambiguous

(d) Both $C_1$ & $J_1$ have equivalent LR(k) grammars.

18. A takes two arbitrary DPDA $M_1$ & $M_2$. He wants to determine if $M_1$ & $M_2$ accept some strings in common.

(a) A has to check for all strings of length $\leq 10$

(b) A has to check for all strings of length $\leq 2^{10}$

(c) A has to check for all strings of length $\leq 2^{10}$

(d) A can never determine if $M_1$ & $M_2$ accept some string in common.

19. A is given a software package $S_1$. To test it exhaustively he generates test data using a form counter machine. Choose the correct statement.

(a) By generating enough test data, $S_1$ can be exhaustively tested, so it is a decidable problem.

(b) Exhaustive testing is an undecidable problem.

(c) Though exhaustive testing is undecidable we can use enough test data to see if that $S_1$ contains < 10 bugs.

keralanotes.com

(d) none of the above.

20. A argues that as the membership problem of regular sets is decidable, if we form the infinite union of regular sets we should decide the membership problem. Choose the correct statement.

(a) A's statement is always true

(b) A's statement is always false

(c) A' s statement is never true for any particular choice of regular sets.

(d) none of the above

21. A is given a R.E. set $L_1$ accepted by some Turing Machine $M_1$. He is also given that $L_1$ is a R.E. set that is not recursive.

(a) $\overline{L_1}$ is always decidable

(b) $\overline{L_1}$ is partially decidable

(c) $\overline{L_1}$ is undecidable

(d) $\overline{L_1}$ can be accepted by the finite union of 1000 counter machines

22. A is given a non- deterministic Turing machine $M_1$ that accepts a language $L_1$ & a deterministic Turing machine accepting a language $L_2$. Choose the correct statement.

(a) As any nondeterministic Turing machine has an equivalent deterministic Turing machine $L_1 = L_2$

(b) If $M_2$ is obtained by replacing the non determinism on $M_1$ by determinism then the time complexity of $L_2$ is polynomial in the size of the input always.

(c) $L_1 = \varphi$ may be undecidable, but $L_2 = \varphi$ is always decidable if h ($L_1$) = $L_2$.

(d) Both $L_1$ & $L_2$ suffer from the halting problem.

keralanotes.com

23. A is given languages $L_1, L_2, L_3, L_4$ & $L_s$; which are a R.E. set, a CSL, a CFL and a regular set. He is also given an $\varepsilon$-free homomorphism h. Choose the correct answer.

(a) $h(L_1)$, $h(L_2)$, $h(L_3)$, $h(L_4)$ & $h(L_{56})$, are regular sets & so their emptiness problem is decidable.

(b) $h(L_1)$, $h(L_2)$, $h^{-1}(L_3)$, $h^{-1}(L_4)$,& $h^{-1}(L_5)$), all are CFLs and so their equivalence problem is decidable for any pair of languages.

(c) $h^{-1}$ $(h(L_1))$, $h^{-1}$ $(h(L_2))$, $h^{-1}$ $(h(L_3))$, $h^{-1}$ $(h(L_4))$,& $h^{-1}$ $(h(L_5))$, are all CSLs or recursive sets and so the problem of whether they are pairwise disjoint is decidable.

(d) The membership problem of $L_1$ is undecidable but the membership problem of $L_2, L_3, L_4$ **&** $L_5$ are undecidable.

**Solutions**

**Level -1**

**01. Ans (b)**

**Sol :** To evaluate we need a turing machine. A stack can only parse it.

**02. Ans (c)**

**Sol :** P is closed under complement, If NP is closed under complement then P = NP. If complement of P = NP then P = NP

**03. Ans (d)**

**Sol :** $\varepsilon$ cannot be in any recursive set of (a), (b) & (c) are excluded.

**04. Ans (d)**

**Sol :** $L_1$ cannot be decided in a finite time, so it is R.E. $L_2$ is simple we run the TM for 1000 steps. So $L_2$ is recursive.

**05. Ans (a)**

keralanotes.com

**Sol :** The given grammar is a DCFL & hence DPDA can accept It. It is the language of palindrome with 'd' in the center. Every DCFL is accepted by a DPDA. A DPDA can be simulated by a PDA. A PDA can be simulated by an LBA. An LBA can be simulated by a TM.

**06. Ans (a)**

**Sol :** The language is a DCFL & hence can be accepted by a DPDA. Hence by a PDA. Hence by a LBA. Hence by a TM

**07. Ans (c)**

**Sol :** A LBA is nothing, but a 2 DFA with an unlimited amount of ink.

**08. Ans (d)**

**Sol :** A finite set is any way accepted by a FA.

**09. Ans (b) & (c)**

**Sol :** The class of CSLs & recursive sets are closed under complement.

**10. Ans (c)**

**Sol :** The string must end with a 0. The 1 takes it back to $q_0$& hence we take $0^n10^n$.

**11. Ans (a)**

**Sol :** The language is a regular set & all the answers in a way are correct. (a) is the tightest & closet answer.

**12. Ans (d)**

**Sol :** The class of finite automata can be put into a one to one correspondence with the integer.

**13. Ans (b)**

**Sol :** P is closed under complement. If the class NP is closed under complement than P = NP.

keralanotes.com

**14. Ans (d)**

**15. Ans (d)**

**Sol :** The subsets of $\sum^*$ are CSLs, CFLs, regular sets etc and all are the formal languages & hence are uncountable.

**16. Ans (d)**

**Sol :** $L = \sum^*$ is not decidable for CFLs.

**17. Ans (b)**

**Sol :** The union of two countable infinite sets is countable infinite.

**18. Ans (b)**

**Sol :** (a) Since $L_1$ & $L_2$ may be accepted by two different finite automata, they may be different finite automata, they may be different regular sets. So it is not true that $L_1$ is always the same as $L_2$.

(b) The equivalence problem of regular sets is decidable. So this is a true statement.

(c) The emptiness problem of regular sets is decidable. For the algorithm just check strings of length 0 to number of states of the minimal DFA accepting the language. If any one of them is accepted then the regular set is not empty. So this is a false statement.

(d) For a counter example consider $L_1 = L_2 = \sum^*$. Then $L_1 \cap L_2 = \sum^*$ is not a finite set. So this is a false statement.

**19. Ans (a)**

Sol : (a) The intersection of two DCFLs is empty is the same as the halting problem of Turing machines and so is undecidable. So this is a true statement.

(b) $L_1$ & $L_2$ may be different DCFLs. So this is a false statement.

(c) $L_2$ need not be a regular set always. So $L_2 = L_3$ is false.

keralanotes.com

(d) It is decidable if a DCFL $L_1$ is equal to a given regular set $L_3$. So this is false.

20. **Ans (a)**

Sol : Two CFLs & the invalid computation of Turing machine can be described by a regular set. So this is undecidable. So (b) is a false statement.

(c) It is decidable if a CFL is finite. Construct the graph of the (CFG) grammar generation $L_1$. If it has no cycles the $L_1$ is finite. So (c) is a false statement.

(d) It is decidable if a CFL is infinite. If the graph of the reduced CFG generating $L_1$.

(a) The emptiness problem of CFG's is decidable. Construct the reduced grammar generating $L_1$; if it vanishes then $L_1$ is empty. So (a) is a true statement.

(b) It is undecidable for a CFL $L_1$, $L_1 = \sum^*$. This follows from the fast that the valid computations of a Turing machine can be given by the intersection of a cycle then $L_1$ is infinite. So (b) is a false statement.

21. **Ans (d)**

**Sol :** Both $C_1$ & $J_2$ are procedure which can realise all the R.E. Sets. So neither $C_1$ nor $J_2$ is necessarily an algorithm.

22. **Ans (b)**

**Sol :** (a) Whether 'a' is ever executed reduces the halting problem of Turing machines & so is undecidable.

(b) Whether 'a' is live or dead reduces the hating problem of turing machine

(c) It is undecidable if C contains a repeated iteration.

23. **Ans (d)**

**Sol :** (a) The graph coloring problem is NP – complete problem which is

keralanotes.com

Perhaps intractable but always decidable as there exists an algorithm of exponential complexity for the problem.

(b) Register optimization can be resolved by the graph coloring problem.

(c) If we have a finite number of nodes then a simple polynomial algorithm suffices for graph coloring & so it is intractable

(d) The graph coloring problem has an algorithm of less than or equal to exponential complexity.

**24. Ans : (c)**

**Sol:** If L is recursive set h(L) will be in general R.E set. So for context sensitive language & recursive sets we have to use $\varepsilon$- free homomorphism in closure properties.

**25. Ans : (d)**

**Sol:** (a) $M_1$ $M_2$ and $M_3$ describe all the R.E. sets and so the halting problem is undecidable.

(b) The equivalence problem of R.E. sets is undecidable
(c) It is undecidable if a turing machine accepts all the strings over the terminal alphabet.
(d) $M_3$ can realise a universal turing machine & so can simulate any turing machine.

**26. Ans : (d)**

**Sol:** $M_1$ & $M_2$ are just variations of the standard model of the turing machine & accept all the R.E. sets & nothing else.

**27. Ans : (c)**

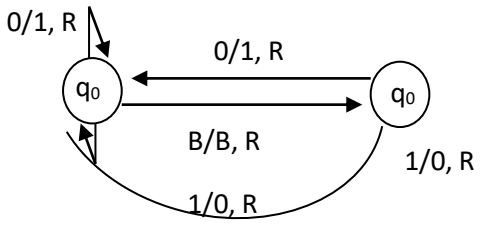
**Sol:** By Rice's Theorem all non-trivial properties of R.E. sets are undecidable. The (c) is a trivial property as there are an infinite number of graphs which have the Hamiltonian cycle.

keralanotes.com

**Level –** 2

**01.** Ans : (a)

**Sol :**




1) ε as input – m/c gets into an infinite loop. So (d) is not a correct choice.

2) 0 as input – accept

**02.** Ans : (d)

**Sol :** S generates $\{a^n b^n c^n / n, m \geq 1\}$

**03.** Ans : (c)

**Sol :** The DCFLs are not closed under intersection. The CFLs are not closed under intersection. The regular sets are closed under intersection.

**04.** Ans : (a)

**Sol :** (a) Regularly is decidable

(c) Membership of R.E. is decidable

**05.** Ans : (d)

**Sol :** $L_1 = \overline{L_n} = \sum^*$ & $L_{n=\varphi}$

**06.** Ans : (a)

**Sol :** It is generally accepted that (a) is the true case

**07.** Ans : (c)

**Sol :** Any automata has a halt state. A TM is not a FA.

**08.** Ans : (d)

**Sol :** The regular expression denotes regular sets.

**09.** Ans : (c)

**Sol :** ε is not accepted by the TM

**10.** Ans : (b)

keralanotes.com

**Sol :** The TM halts so it accepts a recursive set.

**11.** Ans : (a)

**Sol :** $L_1 = (0+1)^*$ is a regular & infinite set by itself.

**12.** Ans : (d)

**Sol :** $L_2$ can be any language.

**13.** Ans : (d)

**Sol :** $\varepsilon$ is accepted & so in 'aa'. Use the methods of elimination

**14.** Ans : (a)

**Sol :** **(a)** The intersection of two DCFLs gives the valid computations of a Turing Machine which corresponds to solutions to the PCP problem P. So this is true.

(b) The intersection of two regular sets is regular. For a regular set we have a FA M accepting it. The halting problem of M is decidable. So this is a false statement.

**15.** Ans : (a)

**Sol :** (a) A CSL can be obtained from the intersection of two CFLs, as the Boolen closured two CFLs are the CSLs. Now the intersection of two CFLs give the valid computations of a Turing Machine & so the given choice reduces to the halting problem which is undecidable. So (a) is ture statement.

(b) The CSLs are automatically recursive sets, so their membership problem is decidable. So this is a false statement.

(c) The equivalence problem of two CFLs is undecidable, so this gives that the equivalence problem of CFLs is undecidable. So this is a false statement.

(d) It is undecidable if a CFL is regular, so this is undecidable.

**16.** Ans : (c)

**Sol :** (a) $L_1$ is recursive set & $L_2$ is R.E. set. The membership problem of $L_2$ is the halting problem of TMs & so undecidable. So this is a false statement.

(b) As $L_1$ is recursive set for which the halting problem is decidable & $L_2$ is a R.E. set for which the halting problem is undecidable, this is a false statement.

(c) It is undecidable if $L_1 = \phi$. So it is undecidable if $L_1 = \sum^*$. $L_1$ is a R.E. set. However some $L_1$ & $L_2$ may be $\sum^*$. So this is true.

(d) It is undecidable if recursive sets and R.E. sets are regular. So this is a false statement.

**17. Ans : (b)**

**Sol : (a)** $C_1$ & $J_1$ generate all C & JAVA programs which are procedures & not algorithms.

(b) It if undecidable if a CFG $C_1$ is ambiguous

(c) It is undecidable if a cfg $J_2$ is ambiguous

(d) Both $C_1$ & $J_1$ deal with CFG's. All CFG's do not have LR(k) equivalents.

**18. Ans : (d)**

**Sol :** The intersection of two DCFLs can give all the valid computations of a turing machine & so is undecidable.

**19. Ans : (b)**

**Sol :** Exhaustive testing reduce to the halting problem of turing machine. So we can never decide if $S_1$ will ever halt.

**20. Ans : (a)**

**Sol :** (a) The infinite union of regular sets can yield any formal languages including undecidable once.

(b) If we choose all regular sets as $\{\varepsilon\}$ then Lavanya's statement is true.

(c) If we choose all regular sets as if then Lavanya's statement is true.

**21. Ans : (c)**

**Sol :** As the R.E. sets are not closed under complement, $\overline{L}_1$ cannot be accepted by a Turing machine. A multi counter Turing machine is just a variation of the standard model of a Turing machine.

**22. Ans : (d)**

**Sol :** (a) $M_1$ & $M_2$ may be machine generating different R.E. sets

(b) Normally removing non-determinism leads to an exponential blown-up in the time complexity.

(c) The emptiness problem of $L_1$ and $L_2$ are undecidable.

(d) Both $M_2$ & $M_1$ can be corrected to the standard model of turing machines.

**23. Ans : (d)**

keralanotes.com

**Sol :** By definition recursive sets have a decidable membership problem. The membership of R.E. sets is undecidable are closed under $\varepsilon$- free homomorphism & so we use the standard results

keralanotes.com