

ACADEMIC PLANNER

For

“AUTOMATA THEORY AND COMPILER DESIGN ”

Presented by

LAL BAHADUR PANDEY

Department of
CSE(AI&ML)



CMR ENGINEERING COLLEGE

(Approved by AICTE-NewDelhi, Affiliated to J.N.T.U, Hyderabad)

Kandlakoya(v),Medchal Road,Hyderabad-501 401,Telangana State, India .Website: www.cmrec.ac.in

(AY:2025-26)

ACADEMIC PLANNER

Subject: AUTOMATA THEORY AND COMPILER DESIGN

<u>S.NO</u>	<u>CONTENT</u>
(1) -	Preamble/Introduction
(2) -	Prerequisites
(3) -	Objectives and Outcomes
(4) -	Syllabus 1.R20-CMREC 2.GATE 3.IES
(5) -	List of Expert Details (Local/National/International with Contact details/Profile link/Blogs/their research Contribution towards the subject)
(6) -	Journals with min 5 ref paper for literature study
(7) -	Subject -Lesson plan
(8) -	Suggested Books (Prescribed and References)
(9) -	Websites for self-learning resources like
(10)	Question Banks 1.JNTUH/Model papers 2.GATE
(11) -	Two case study presentations with Project / Product/ Model /prototypes/ Industrial applications
(12) -	Assignment Question/Innovative Assignments sets.
(13) -	List of topics for students Seminars with Guidelines
(14) -	STEP/Course material in softcopy
(15) -	Expert Lectures with topics &Schedules(if any)

1. Preamble/Introduction:

It is the study of abstract computing device. Theory of computation is the elementary ways in which a computer can be made to think. It is strong foundation for a lot of abstract areas of computer science. Theory of computation is a theoretical part of computer science. A compiler is a translator that converts the high-level language into the machine language. High-level language is written by a developer and machine language can be understood by the processor. Compiler is used to show errors to the programmer. The main purpose of compiler is to change the code written in one language without changing the meaning of the program. When you execute a program which is written in HLL programming language then it executes into two parts. In the first part, the source program compiled and translated into the object program (low level language). In the second part, object program translated into the target program through the assembler.

2.PREREQUISITES

A course on “Programming for Problem Solving”

A course on “Computer Organization and architecture”

3. OBJECTIVES & OUTCOMES

1. Objectives

1. To provide introduction to some of the central ideas of theoretical computer science from the perspective of formal languages.
2. To introduce the fundamental concepts of formal languages, grammars and automata theory.
3. Classify machines by their power to recognize languages.
4. Employ finite state machines to solve problems in computing.
5. To understand deterministic and non-deterministic machines.
6. To understand the differences between decidability and undecidability.
7. Introduce the major concepts of language translation and
Compiler design and impart the knowledge of practical skills necessary for constructing a compiler.
8. Topics include phases of compiler, parsing, syntax directed translation, Type checking use of symbol tables, code optimization techniques, intermediates code generation, code generation and data flow analysis

2. Outcomes

CO1	Student will be able to Design NFA & DFA and check acceptability of generating a regular language;
CO2	Student will be able to Demonstrate the language accepted by automata or generated by a regular expression or a context-free grammar and Explain automata and context-free grammar to determine certain word belongs to a language.
CO3	Student will be able to Design solution of complex problems using PDA and Turing Machine, and check the decidability of problems.
CO4	Student will be able to Explain structure of compiler, Design and implement TOP-down parsing and bottom-up parsing technics.

CO5	Student will be able to Write SDD, SDT & Apply for Intermediate-Code Generation technics.
------------	---

4. SYLLABUS

4.1 CMREC Autonomous Syllabus(R-22)

UNIT - I

Introduction to Finite Automata: Structural Representations, Automata and Complexity, the Central Concepts of Automata Theory – Alphabets, Strings, Languages, Problems.

Nondeterministic Finite Automata: Formal Definition, an application, Text Search, Finite Automata with Epsilon-Transitions.

Deterministic Finite Automata: Definition of DFA, How A DFA Process Strings, The language of DFA, Conversion of NFA with ϵ -transitions to NFA without ϵ -transitions. Conversion of NFA to DFA

UNIT - II

Regular Expressions: Finite Automata and Regular Expressions, Applications of Regular Expressions, Algebraic Laws for Regular Expressions, Conversion of Finite Automata to Regular Expressions.

Pumping Lemma for Regular Languages: Statement of the pumping lemma, Applications of the Pumping Lemma.

Context-Free Grammars: Definition of Context-Free Grammars, Derivations Using a Grammar, Leftmost and Rightmost Derivations, the Language of a Grammar, Parse Trees, Ambiguity in Grammars and Languages.

UNIT - III

Push Down Automata: Definition of the Pushdown Automaton, the Languages of a PDA, Equivalence of PDA's and CFG's, Acceptance by final state.

Turing Machines: Introduction to Turing Machine, Formal Description, Instantaneous description, The language of a Turing machine.

Undecidability: Undecidability, A Language that is Not Recursively Enumerable, An Undecidable Problem That is RE, Undecidable Problems about Turing Machines.

UNIT – IV

Introduction: The structure of a compiler, Lexical Analysis: The Role of the Lexical Analyzer, Input Buffering, Recognition of Tokens, The Lexical- Analyzer Generator Lex, Syntax Analysis: Introduction, Context-Free Grammars, Writing a Grammar, Top-Down Parsing, Bottom- Up Parsing, Introduction to LR Parsing: Simple LR, More Powerful LR Parsers.

UNIT - V

Syntax-Directed Translation: Syntax-Directed Definitions, Evaluation Orders for SDD's, Syntax-Directed Translation Schemes, Implementing L-Attributed SDD's.

Intermediate-Code Generation: Variants of Syntax Trees, Three-Address Code

Run-Time Environments: Stack Allocation of Space, Access to Nonlocal Data on the Stack, Heap Management.

4.2 Syllabus Gate and PSU's: -

Regular Expression & finite Automata, Context free grammar and push down automata, Regular and context free language, pumping lemma, Turing machine and undecidability. Lexical analysis, parsing, syntax directed translation, run time environment, intermediate code generation, Local optimization, Data flow analysis, constant propagation, Liveness analysis, Common subexpression elimination. (10 question of Gate exam)

5. Expert details

List of expert details (local/national/international with contact details/profile link/blogs/their research contribution towards the subject)

- **Local:**

Dr. Ashok Kumar Das

Professor of Computer Science, Center for Security, Theory and Algorithmic Research, IIIT Hyderabad

<https://www.iiit.ac.in/people/faculty/ashokkdas/>

Dr. Raghu kishore neelishetti,

Mahindra University, Hyderabad, India.

<https://www.mahindraecolecentrale.edu.in/faculty/raghu-kisore-neelisetti>

- National:

Dr. Sushanta karmakar ,IIT Guwahati, India. <https://www.iitg.ac.in/sushantak/>

Dr. Bruhdeswar Bezwada, IITS jammu , india.

<https://iitjammu.ac.in/faculty/~bruhadeshwarbezwada>

- international:

Dr. kouichi sakurai, kyushu university, japan. <https://hyoka.ofc.kyushu-u.ac.jp/search/details/k000220/english.html>

6. JOURNALS

1. Theory of Computing: An Open Access Electronic Journal: dedicated to free global dissemination of research in theoretical **computer science**

<http://theoryofcomputing.org/index.html>

2. IEEE Transactions on Computers: This will help in design of **algorithmic research** in this broad sense. The subtitle **Cognition, Informatics, and Logic** emphasizes the intended breadth and interdisciplinary nature of the journal.

<https://ieeexplore.ieee.org/document/9387475>

3. Theory of Computing Systems (TOCS) is devoted to publishing original research from all areas of **theoretical computer science**, ranging from foundational areas such as computational complexity, to fundamental areas such as algorithms and data structures, to focused areas such as parallel and distributed algorithms and architectures.

<https://www.springer.com/journal/224>

4. Algorithms: MDPI (ISSN 1999-4893) is a [peer-reviewed](#), open access

journal which provides an advanced forum for studies related to algorithms and their applications. *Algorithms* is published monthly online by MDPI.

<https://www.mdpi.com/journal/algorithms>

5. “Energy-Efficient Backend Compiler Design for Embedded System” by Wen-Tsong Shiue, Member, IEEE.
<https://ieeexplore.ieee.org/abstract/document/949560>
6. “Formal compiler construction in a logical framework” by Jason Hickey & Aleksey Nogin.
<https://link.springer.com/article/10.1007/s10990-006-8746-6>
7. “An empirical characterization of stream programs and its implications for language and compiler design” by William Thies, Saman Amarasinghe.
<https://dl.acm.org/doi/abs/10.1145/1854273.1854319>
8. “Programming languages and compiler design for realistic quantum hardware” By Frederic T. Chong, Diana Franklin & Margaret Martonosi.
<https://www.nature.com/articles/nature23459>

Subject Lesson Plan (CMREC -R(22))

S.NO	Topic (syllabus)	NO. OF LECTURES REQUIRED	Suggested Books	Teaching Methods
	Unit – 1			
1	Introduction to Finite Automata: Structural Representations, Automata and Complexity, the Central	2	T1,T3,R1	M1
2	Concepts of Automata Theory–Alphabets, Strings, Languages, Problems.	3	T1, T3,R1	M1
3	Nondeterministic Finite Automata: Formal Definition, an application, Text Search, Finite Automata with Epsilon-Transitions.	2	T1,T3,R2	M1
4	Deterministic Finite Automata: Definition of DFA, How A DFA Process Strings,	2	T1,T3,R3	M1
5	The language of DFA, Conversion of NFA with ϵ -transitions to	2	T1,T3,R4	M1

	NFA without ϵ -transitions.			
6	Conversion of NFA to DFA	2	T1,R1	M1
	UNIT -2			
7	Regular Expressions: Finite Automata and Regular Expressions	1	T1,R2,T3	M1
8	Applications of Regular Expressions, Algebraic Laws for Regular Expressions,	2	T1,R5	M1 & M2
9	Conversion of Finite Automata to Regular Expressions.	2	T1,T3	M1
10	Pumping Lemma for Regular Languages: Statement of the pumping lemma, Applications of the Pumping Lemma.	2	T1,T3	M1
11	Context-Free Grammars: Definition of Context-Free Grammars, Derivations Using a Grammar, Leftmost and Rightmost Derivations	2	T1,T3,R1	M1
12	, the Language of a Grammar, Parse Trees, Ambiguity in Grammars and Languages.	2	T1,T3,R3	M1 & M2
	UNIT-3			
13	Push Down Automata: Definition of the Pushdown Automaton, the Languages of a PDA	2	T1,T3,R1	M1
14	Equivalence of PDA's and CFG's, Acceptance by final state.	2	T1,T3,R2	M1
15	Turing Machines: Introduction to Turing Machine, Formal	3	T1,T3,R3	M1 & M2

	Description, Instantaneous description, The language of a Turing machine.			
16	Undecidability: Undecidability, A Language that is Not Recursively Enumerable	2	T1,T3,R4	M1
17	An Undecidable Problem That is RE, Undecidable Problems about Turing Machines	2	T1,T3,R5	M1 & M2
	UNIT-4			
18	Introduction: The structure of a compiler, Lexical Analysis: The Role of the Lexical Analyzer	2	T2,R6	M1 & M2
19	Input Buffering, Recognition of Tokens, The Lexical-Analyzer Generator Lex,	1	T2,R6	M1 & M2
20	Syntax Analysis: Introduction, Context-Free Grammars, Writing a Grammar,	2	T2,R7	M1
21	Top-Down Parsing	2	T2,R6,R7	M1
22	Bottom- Up Parsing, Introduction to LR Parsing: Simple LR, More Powerful LR Parsers.	3	T2,R7	M1
	UNIT-5			
24	Syntax-Directed Translation: Syntax-Directed Definitions, Evaluation Orders for SDD's,	2	T2,R6	M1 & M2
25	Syntax-Directed Translation Schemes, Implementing L-Attributed SDD's.	1	T2,R7	M1 & M2
26	Intermediate-Code Generation: Variants of Syntax Trees, Three-Address Code	2	T2,R6	M1 & M2
27	Run-Time Environments: Stack Allocation of Space	1	T2,R7	M1 & M2

28	Access to Nonlocal Data on the Stack, Heap Management.	2	T2,R7	M1 & M2
----	--	---	-------	---------

M1- White Board and Markar

M2- ICT METHOD (PPT/E-Resources/NPTEL)

8. SUGGESTED BOOKS

TEXT BOOKS:

1. Introduction to Automata Theory, Languages, and Computation, 3rd Edition, John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, Pearson Education.
2. Compilers: Principles, Techniques and Tools, Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, 2nd Edition, Pearson.
3. Theory of Computer Science – Automata languages and computation, Mishra and Chandrasekaran, 2nd Edition, PHI

REFERENCE BOOKS:

1. Introduction to Languages and The Theory of Computation, John C Martin, TMH.
2. Introduction to Computer Theory, Daniel I.A. Cohen, John Wiley.
3. A Text book on Automata Theory, P. K. Srimani, Nasir S. F. B, Cambridge University Press.
4. Introduction to the Theory of Computation, Michael Sipser, 3rd edition, Cengage Learning.
5. Introduction to Formal languages Automata Theory and Computation Kamala Krithivasan, Rama R, Pearson.
6. Lex & Yacc – John R. Levine, Tony Mason, Doug Brown, O'reilly
7. Compiler Construction, Loudon, Thomson.

9.WEBSITES For Self Learning Resources

- a. <https://nptel.ac.in/courses/106/105/106105191/>
- b. https://www.tutorialspoint.com/computer_concepts/index.htm
- c. https://www.w3schools.com/ai/ai_history_computers.asp
- d. <https://ocw.mit.edu/courses/18-404j-theory-of-computation-fall-2020/>
- e. <https://www.geeksforgeeks.org/introduction-of-theory-of-computation/>
- f. <https://www.geeksforgeeks.org/theory-of-computation-automata-tutorials/>
- g. <https://www.javatpoint.com/automata-tutorial>

10. QUESTION BANKS

- 1) Define Finite Automata.
- 2) Define Context Free Grammar.
- 3) What do you mean by Context Sensitive Language?
- 4) Define Pushdown Automata.
- 5) Give an example of an ambiguous grammar.
- 6) Design a Turing Machine to recognize a string consisting of an Even number of 1's.
- 7) Define Chomsky Normal Form.
- 8) Explain rewriting systems.
- 9) Explain the difference between the transition functions of DFA and NFA.
- 10) If u and v are Regular expression $ababbadbbaa$ then find uv and u^*v .
- 11) Construct Finite Automata equivalent to the regular Expression $(0 + i)^* (00 + 11) (0 + 1)^*$.

Design a Turing Machine over $\{1, b\}$ which can compute a concatenation function over $\Sigma = \{1\}$. If a pair of words (w_1, w_2) is the input, the output has to be $w_1 w_2$.

- 13) Explain the properties of LL (k) and LR (k) grammars.
- 14) What is context-free grammar?
- 15) What is Griebach normal form?
- 16) What do you understand by type-1 grammar?
- 17) What are palindromes?
- 18) What do you understand by acceptability machine?
- 19) What is type-2 grammar?
- 20) Define LR(k) grammar.
- 21) Why do natural languages are not formal languages?
- 22) What do you understandability term union of sets?
- 23). Define Token, Pattern and Lexeme

24).Eliminate left recursion from the following grammar

$E \rightarrow E+T/T$

$T \rightarrow T*F/F$

$F \rightarrow (E)/id$

25).Define Input Buffering?

26). Differentiate between top down and bottom-up parsing techniques.

27). Define compiler, interpreter & assembler?

28) .Eliminate left recursion from the following grammar

$S \rightarrow (L)/a$

$L \rightarrow L,S/S$

29) .Write difference between one pass & two pass compiler ?

30. Write role of lexical analyzer with diagram?

31. Define Syntax Directed Definition with example

32 .Draw NFA for $R = (a/b)^*abb$

33. Write limitation of TOP Down parsing?

34. Find left most derivation for $w = a+b*a+b$ using following grammar.

$E \rightarrow E+E/E*E/E-E/a/b$

35 .Explain the various phases of a compiler with example also write input and output of each phases?

36.Construct the predictive parserLL(1) table for the following grammar

$E \rightarrow TE' | T$

$E' \rightarrow +TE' | \epsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' | \epsilon$

$F \rightarrow (E)/id$

37 .Construct SLR(1) parsing table for the given grammar

$S \rightarrow L=R/R$

$R \rightarrow L$

$L \rightarrow *R/id$

38 .Construct LALR(1) parsing table for the given grammar

$S \rightarrow AA$

A \rightarrow aA/b

39. Construct the predictive parser LL(1) table for the following grammar & parse the string w=(a,a) using given grammar?

S \rightarrow (L)/a

L \rightarrow L,S/S

40. Construct the predictive parser LL(1) table for the following grammar

S \rightarrow iEtS/iEtses/a

E \rightarrow b

41. Show that given grammar is SLR(1) but not LL(1)

S \rightarrow SA/A

A \rightarrow a

42. Explain the role of intermediate code generator in compilation process.

43. Define Syntax Directed Translation with example

44. What is a DAG? Explain with example

45. Define constant folding.

46. Explain about code motion.

47. Write a short note on copy Propagation.

48. Explain intermediate code representations and its types.

49. What is activation records.

50. Explain the role of code generator in compilation process

51. Explain Loops in Flow Graph with example

52. Construct the following expression: $(a + b) * (c + d) + (a + b + c)$ into

a) Quadruples b) Triples c) Indirect Triples

53. Explain various storage allocation strategies with examples

54. Construct a Quadruple, Triples & Indirect Triples for the following expression: $a + a * (b - c) + (b - c) * d$?

55. What is a three-address code? Mention its types. How would you implement the three address statements? Explain with examples

56. Construct the following expression: $(x + y) * (y + z) + (x + y + z)$ into

a) Quadruples b) Triples c) Indirect Triples

57. Explain various storage allocation strategies with examples

58. Explain various method to handle peephole optimization
59. Explain various code optimization techniques in detail
60. Explain static and stack storage allocations?
61. Explain Activation Records with diagram
62. Write short notes on – a) constant propagation b) Partial redundancy elimination with example.



FLAT QNS (2).pdf

Assignment Question Set

SET-1

1. (a) Define NFA and explain with an example.
(b) Conclude what type of strings will be accepted by the below
Finite automata as shown in figure 1b. [6+10]

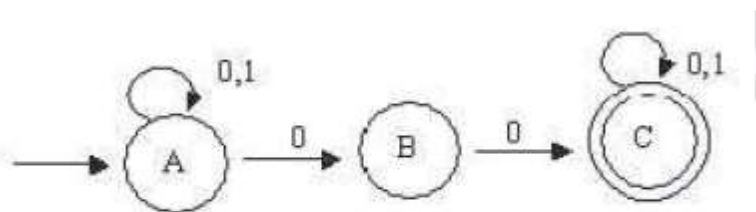


Figure 1b

2. (a) Design a Moore Machine to determine the residue mod 4 for each binary string treated as integer.
(b) Design a Mealy machine that uses its state to remember the last symbol read and emits output 'y' whenever current input matches to previous one, and emits n otherwise.
3. Write a note on Formal Languages and grammars.
4. Construct DFA which accepts strings having odd number of a's and even number of b's.
5. Design a Turing machine M to recognize the language

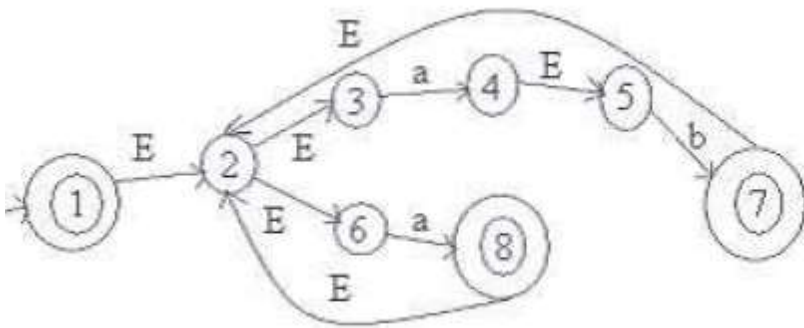
$$\{a^n b^n \mid n \geq 1\}$$
6. Prove

$$(1 + 00^*1)^+ (1 + 00^*1)(0 + 10^*1)^* (0 + 10^*1) = 0^*1 (0 + 10^*1)^* .$$
7. Define pushdown automata completely.'

8. Design a Turing machine over $\{1, b\}$ which can compute concatenation function over $I = \{1\}$. If a pair of words (w_1, w_2) is an input, the output has to be w_1w_2 .
9. Prove that grammar $S \rightarrow 0A2, A \rightarrow 1A1, A \rightarrow 1$ is not LR(0)

SET-2

1. For the following NFA with 2-moves convert it into an NFA without 2-moves and show that NFA with 2-moves accepts the same language as shown in figure 2.



2. (a) When is a grammar said to be in reduced form.

(b) Convert the following grammar to GNF:

$$G = (\{A_1, A_2, A_3\}, \{a, b\}, P_1, A_1)$$

Where P consists of the following:

$$A_1 \rightarrow A_2A_3$$

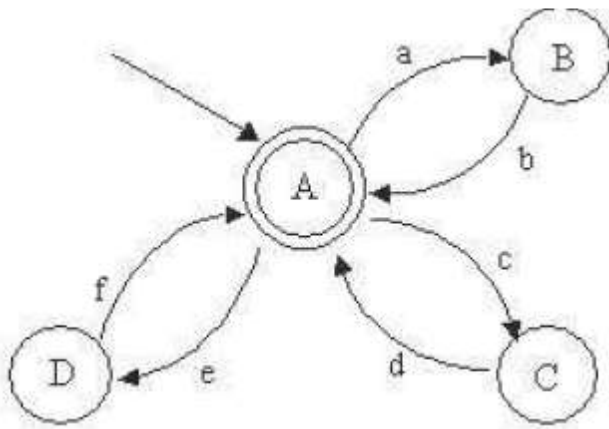
$$A_2 \rightarrow A_3A_1/b$$

$$A_3 \rightarrow A_1A_2/a. [8+8]$$

3. (a) Define PDA. In what ways a PDA can show the acceptance of a string. Explain with examples.

(b) Construct the PDA M for the language $L = \{ww^R/w \in \{a, b\}^*\}$ such that $sL = L(M)$.

7. Out of the following languages, which are/is accepted by given FA and explain as shown in figure 1.



- (a) $(a+b)^* (c+d)^* (ef)^*$
 (b) $(ab)^* (cd)^* (ef)^*$
 (c) $(a+b)^* + (c+d)^* + (ef)^*$
 (d) $((ab)^* + (cd)^* + (ef)^*)^*$

5) (a) Show that the FA are equivalent as shown in figure 2a.

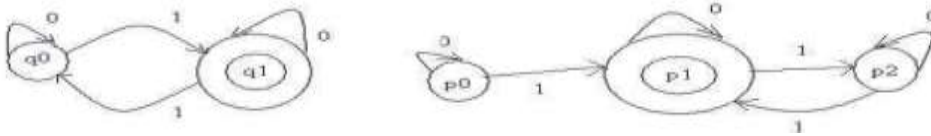


Figure 2a

SET -3

1. Define Token, Pattern and Lexeme

2. Eliminate left recursion from the following grammar

$F \rightarrow E+T/T$

$U \rightarrow T * F/F$

$F \rightarrow (E)/id$

3. Define Input Buffering?

4. Differentiate between top down and bottom-up parsing techniques.

5. Define compiler, interpreter & assembler?

6. Eliminate left recursion from the following grammar

$S \rightarrow (L)/a$

$L \rightarrow L, S/S$

7. Write difference between one pass & two pass compiler ?

8. Write role of lexical analyzer with diagram?

9. Define Syntax Directed Definition with example

10. Draw NFA for $R = (a/b)^*abb$

11. Write limitation of TOP Down parsing?

12. Find left most derivation for $w = a + b * a + b$ using following grammar.

$E \rightarrow E + E / E * E / E - E / a / b$

13. Explain the various phases of a compiler with example also write input and output of each phases?

14. Construct the predictive parser LL(1) table for the following grammar

$E \rightarrow TE' | T$

$E' \rightarrow +TE' | \epsilon$

$T \rightarrow FT'$

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

$F \rightarrow (E) / id$

15. Construct SLR(1) parsing table for the given grammar

$S \rightarrow L = R / R$

$R \rightarrow L$

$L \rightarrow *R / id$

16. Construct LALR(1) parsing table for the given grammar

$S \rightarrow AA$

$B \rightarrow aA / b$

17. Construct the predictive parser LL(1) table for the following grammar & parse the string $w = (a, a)$ using given grammar?

$S \rightarrow (L) / a$

$L \rightarrow L, S / S$

18. Construct the predictive parser(1) table for the following grammar

$S \rightarrow iEtS / iEtses/a$

$E \rightarrow b$

19. Show that given grammar is SLR(1) but not LL(1)

$S \rightarrow SA / A, A \rightarrow a$

(11) Case study presentations with Project / Product/ Model /prototypes/ Industrial applications

- A) Designing of lexical analysis of a compiler.
- B) Recognition of Reserved Words and Identifiers

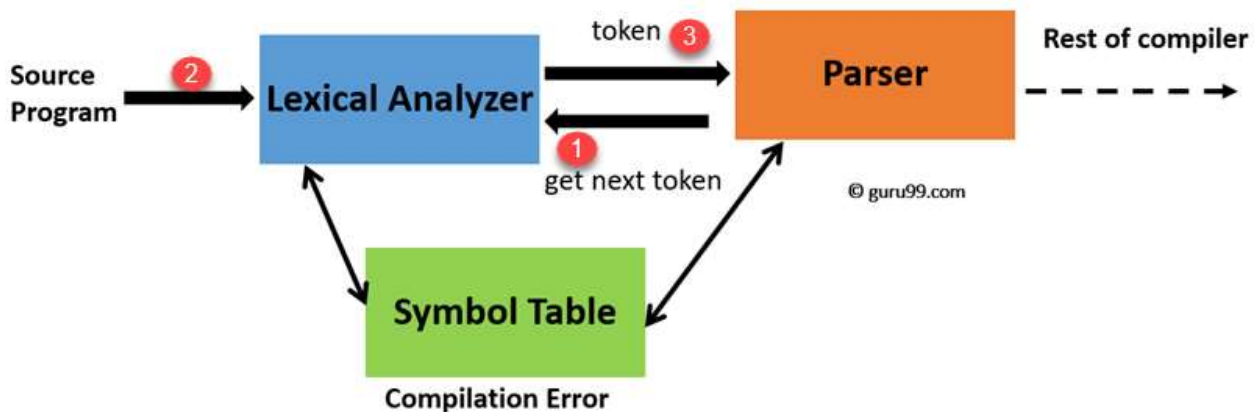
A) **Lexical Analysis** is the very first phase in the compiler designing. A Lexer takes the modified source code which is written in the form of sentences . In other words, it helps you to convert a sequence of characters into a sequence of tokens. The lexical analyzer breaks this syntax into a series of tokens. It removes any extra space or comment written in the source code.

Programs that perform Lexical Analysis in compiler design are called lexical analyzers or lexers. A lexer contains tokenizer or scanner. If the lexical analyzer detects that the token is invalid, it generates an error. The role of Lexical Analyzer in compiler design is to read character streams from the source code, check for legal tokens, and pass the data to the syntax analyzer when it demands.

Lexical Analyzer Architecture: How tokens are recognized

The main task of lexical analysis is to read input characters in the code and produce tokens.

Lexical analyzer scans the entire source code of the program. It identifies each token one by one. Scanners are usually implemented to produce tokens only when requested by a parser. Here is how recognition of tokens in compiler design works-



Lexical Analyzer Architecture

1. "Get next token" is a command which is sent from the parser to the lexical analyzer.
2. On receiving this command, the lexical analyzer scans the input until it finds the next token.
3. It returns the token to Parser.

Lexical Analyzer skips whitespaces and comments while creating these tokens. If any error is present, then Lexical analyzer will correlate that error with the source file and line number.

Recognition of Reserved Words and Identifiers

Recognizing keywords and identifiers presents a problem. Usually, keywords like **if** or **then** are reserved (as they are in our running example), so they are not identifiers even though they *look* like identifiers. Thus, although we typically use a transition diagram like that of Fig. 3.14 to search for identifier lexemes, this diagram will also recognize the keywords **if**, **then**, and **else** of our running example.

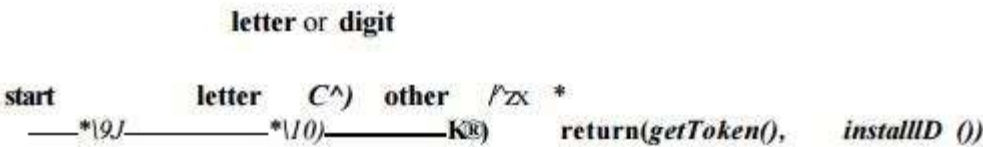
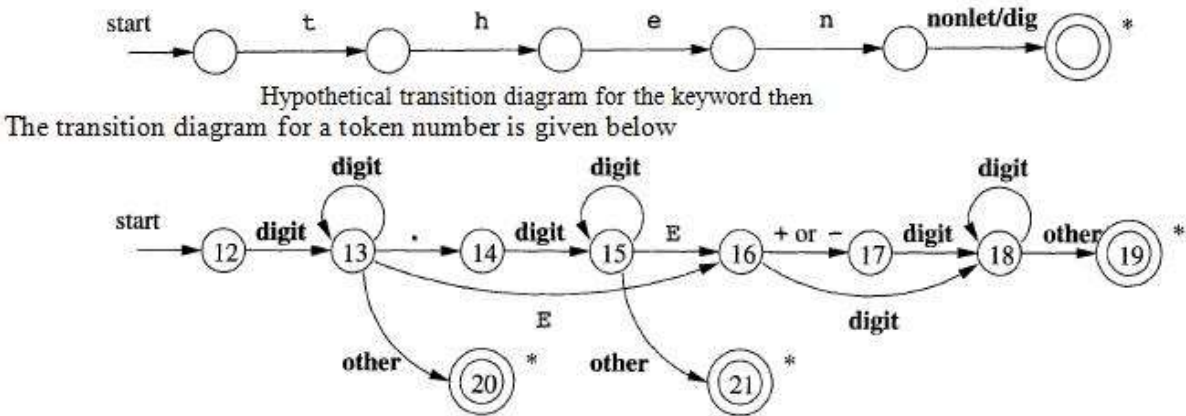


Figure 3.14: A transition diagram for **id**'s and keywords

There are two ways that we can handle reserved words that look like identifiers:

Install the reserved words in the symbol table initially. A field of the symbol-table entry indicates that these strings are never ordinary identi- fiers, and tells which token they represent. We have supposed that this method is in use in Fig. 3.14. When we find an identifier, a call to *installed* places it in the symbol table if it is not already there and returns a pointer to the symbol-table entry for the lexeme found. Of course, any identifier not in the symbol table during lexical analysis cannot be a reserved word, so its token is **id**. The function *getToken* examines the symbol table entry for the lexeme found, and returns whatever token name the symbol table says this lexeme represents — either **id** or one of the keyword tokens that was initially installed in the table.

1. Create separate transition diagrams for each keyword; an example for the keyword **then** is shown in Fig. 3.15. Note that such a transition diagram consists of states representing the situation after each successive letter of the keyword is seen, followed by a test for a "nonletter-or-digit," i.e., any character that cannot be the continuation of an identifier. It is necessary to check that the identifier has ended, or else we would return token **then** in situations where the correct token was **id**, with a lexeme like **then e x t** value that has **then** as a proper prefix. If we adopt this approach, then we must prioritize the tokens so that the reserved-word tokens are recognized in preference to **id**, when the lexeme matches both patterns. We *do not* use this approach in our example, which is why the states in Fig. 3.15 are unnumbered.



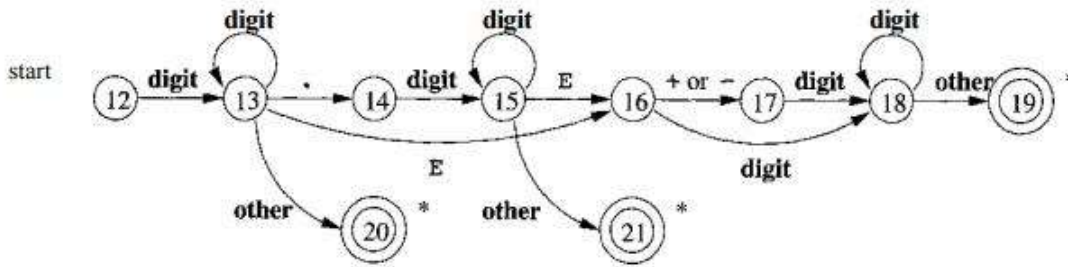


Figure 3.16: A transition diagram for unsigned numbers

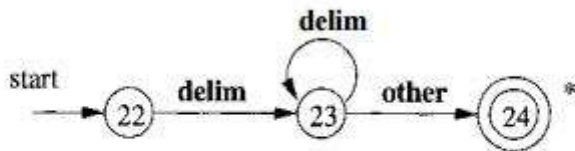


Figure 3.17: A transition diagram for whitespace

(12) TOPICS FOR STUDENT'S SEMINARS

- 1)Regular grammars
- 2)Push down automata
- 3)Greibach normal form
- 4)Pumping Lemma
- 5)Conversion of Finite Automata to Regular expressions
- 6)DCFL and DPDA
- 7)Turing Machine
- 8)Chomsky hierarchy of languages
- 9)P and NP problems
- 10)undecidability of posts. Correspondence problem
11. Intermediate code generation in compilers
12. Code generator for java
13. Code generator wizard eclipse plug in
14. Different Cross compilers gcc cross compiler
15. cross compiler wiki,

14. STEP MATERIAL—



FLAT Unit 1
(2)_merged.pdf



toc-klp-mishra.pdf



cd unit 1_merged
(1).pdf

15. EXPERT LECTURE SCHEDULE:

UNIT	CONTENT / TOPIC DETAILS	DATE
UNIT -1	Introduction to Finite Automata, NFA, DFA	DR. SIDDHARTH SANKAR SHUKLA CSVТУ, BHILAI
UNIT -2	Push down automata Turing machine	DR. SIDDHARTH SANKAR SHUKLA CSVТУ, BHILAI
UNIT -3	Top-down parser	DR. SIDDHARTH SANKAR SHUKLA CSVТУ, BHILAI
UNIT -4	Bottom-up parser LR,SLR,LALR,CLR	Dr. Toran Verma CSVТУ BHILAI (C.G)
UNIT -5	Machine dependent code generation Data flow analysis, Control Flow Analysis	Dr. Toran Verma CSVТУ BHILAI (C.G)

THANK YOU

----- THE END-----