THEORY OF COMPUTATION AND COMPILERS

Unit - I | Lecture- 01

- ✓ Course Objectives
- ✓ Course Outcomes
- ✓ Syllabus
- ✓ Suggested Reading
- ✓ Introduction to Formal Languages and Automata Theory
- ✓ Basic Fundamentals /Central Concepts of Automata Theory

Dr. R. Madana Mohana
Professor, Artificial Intelligence & Data Science | I/c-Head, Artificial Intelligence & Machine Learning
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY
Hyderabad - 500 075, Telangana, INDIA
www.cbit.ac.in

THEORY OF COMPUTATION AND COMPILERS

Course Objectives:

1. Learn the foundations of automata theory, computability theory, and complexity theory. Shows relationship between automata and formal languages.
2. Addresses the issue of which problems can be solved by computational means (decidability vs undecidability)
3. Learn the concepts related to computational complexity of problems.
4. Understand the concept of algorithm and compare the complexity of problems.

THEORY OF COMPUTATION AND COMPILERS

Course Outcomes:

After completion of the course students should able to

1. **Understand** formal language basics and the power of automata to recognize the languages
2. **Analyze** the concept compilation Process and data structures of a compiler
3. **Attains** the knowledge of context free grammars and able to implement parsers
4. **Design** Syntax directed translation scheme for a given Context free grammar and generation of intermediate code
5. **Apply** Optimization to intermediate code and machine code
6. **Illustrate** various object forms, error recovery and tools of a compiler

THEORY OF COMPUTATION AND COMPILERS

Syllabus:

UNIT-I:

Formal Language and Regular Expressions: Chomsky hierarchy, Languages regular expressions, Finite Automata – DFA, NFA. Conversion of regular expression to NFA, NFA to DFA.

Overview of Compilation: Phases, Lexical Analysis, Lex Specifications, Structure of a Lex Specification File, Regular Grammar and Regular Expression for Common Programming Language Features, Pass and Phases of Translation, Interpretation, Bootstrapping, Data Structures of Compiler, LEX tool.

THEORY OF COMPUTATION AND COMPILERS

Syllabus:

UNIT-II:

Context Free grammars and parsing: Context free grammars, derivation, parse trees, ambiguity, Types of Parsers LL(K) grammars and LL(1) parsing.

Bottom-up parsing: Handle pruning LR Grammar Parsing, LALR parsing, parsing ambiguous grammars, Error Recovery in Parsing YACC programming specification.

THEORY OF COMPUTATION AND COMPILERS

Syllabus:

UNIT-III:

Semantic Analysis: Intermediate Forms of Source Programs - Abstract Syntax Tree, Polish Notation and Three Address Codes. Attributed Grammars, Syntax Directed Translation, Language Intermediate Code Forms, Type Checker.

Symbol Table: Symbol Table Format, Organization for Block Structures Languages, Hashing.

THEORY OF COMPUTATION AND COMPILERS

Syllabus:

UNIT-IV:

Code Optimization: Consideration for Optimization, Scope of Optimization, Local Optimization, Loop Optimization, Frequency Reduction, Folding, DAG Representation.

Data Flow Analysis: Flow Graph, Data Flow Equation, Global Optimization, Redundant Sub Expression Elimination, Induction Variable Elements, Live Variable Analysis, Copy Propagation.

THEORY OF COMPUTATION AND COMPILERS

Syllabus:

UNIT-V:

Object Code Generation: Object code forms, machine dependent code optimization, register allocation and assignment generic code generation algorithms.

Error Recovery: various errors in phases and recovery of errors in compilation, introduction to tools of compiler.

THEORY OF COMPUTATION AND COMPILERS

TEXT BOOKS:

1. John E. Hopcroft, Rajeev M & J D Ullman: “Introduction to Automata Theory Languages &Computation”, 3rd Edition, Pearson Education, 2007.
2. Aho, Ullman, Ravisethi: “Compilers Principles, Techniques and Tools”, 2nd Edition, Pearson Education, 2009.

THEORY OF COMPUTATION AND COMPILERS

NPTEL RESOURCES:

1. NOC: Introduction to Automata, Languages and Computation

https://nptel.ac.in/courses/106/105/106105196/

2. Compiler Design – Web Course:

https://nptel.ac.in/courses/106104072

3. NOC: Compiler Design, IIT Kharagpur

https://nptel.ac.in/courses/106105190

4. Compiler Design, IISc Bangalore

https://nptel.ac.in/courses/106108052

OUTLINE

- Introduction to Automata Theory or Theory of Computation
- Applications of Automata Theory or Theory of Computation
- Basic Fundamentals /Central Concepts of Automata Theory
- Formal Languages and Chomsky Hierarchy

INTRODUCTION TO AUTOMATA THEORY / THEORY OF COMPUTATION

Introduction to Automata:

- An **automata** or **automaton** is a machine designed to respond to encoded instructions(for a robot).
- **Automata** is derived from two Greek words '**Auto**' and '**Meta**'.
- **Auto** means one self and **meta** means machine.
- Therefore **Automata** means one self machine.

Introduction to Automata:

General definition of Automata:

- **Automata** is a system where energy, materials and information are transformed, transmitted and used for performing some functions without direct involvement of a human.
- In **Computer Science**, the term **Automata** means **discrete automaton** (Discrete means individual)

Introduction to Automata:

Examples:

- Automatic packing machine
- Automatic machine tools
- Automatic washing machine
- Automatic photo printing machine etc..

Introduction to Automata:

Automata theory or Theory of Computation:

- The theory of computation or automata theory describes the basic ideas and models underlying computing.
- Computation involves taking some inputs and performing the required operations on it using syntactic procedures and algorithms and producing the outputs accordingly.

Introduction to Automata:

Model of Discrete Automata or Automata:

$I_1, I_2, \ldots I_n$ are Inputs

$O_1, O_2, \ldots O_n$ are Outputs

Introduction to Automata:

Automata theory or Theory of Computation:

- The term theory of computation or automata theory suggests various abstract models of computation, represented mathematically.
- Some of these models are as powerful as real time computers. Each abstract computing machine recognizes a **`formal language`**.

APPLICATIONS OF AUTOMATA THEORY OR THEORY OF COMPUTATION

Applications of Automata Theory:

There are several applications of automata theory, some of them are listed below:

1. Switching and Theory and Designing of Digital Circuits
2. Robotics
3. Compiler construction (for 1st phase of compiler i.e., Lexical Analysis or Scanner)
4. Designing of Editors i.e., Text Editors
5. Natural Language Processing (NLP)

Applications of Automata Theory:

6. Knowledge Discovery
7. To verify the correctness of a program
8. To design Finite State Machine (FSM) also called as Finite Automata (FA)
9. Informal methods of design and analysis of complex software and hardware systems.
10. Formal languages and grammars are widely used in connection with programming languages

BASIC FUNDAMENTALS /CENTRAL CONCEPTS OF AUTOMATA THEORY

Basic Fundamentals /Central Concepts of Automata Theory

- Symbol
- Alphabet
- String or Word
- String Operations
- Language
- Language Operations
- Formal Language
- Grammar or Formal Grammar

Symbol

A **`symbol`** is an abstract entity that we shall not define formally.

Example:

Letters: **`A to Z`** (upper case) or **`a to z`** (lower case)

Digits: **`0 to 9`**

Special characters

Alphabet

An **alphabet** is a non-empty finite set of symbols. It is denoted by the symbol **Σ** (sigma)

Examples:

Σ = {a, b, c} is an alphabet which consisting of letters .

Σ = {0, 1} is an alphabet which consisting of digits .

Σ = {a, 0} is an alphabet which consisting of a letter ‘**a**’ and digit **0** .

String or Word

A **string** or **word** is defined as finite sequence of symbols over an alphabet (**Σ**).

Examples:

1. Alphabet **Σ = {a, b}**

Strings:

w = {a, b, aa, ab, ba, bb, aaa, aab, aba, baa, bbb, ….}

2. **Σ = {0, 1}**

w = {0, 1, 00, 01, 10, 11, 000, 001, 010, 100, 111, ….}

String Operations

i) Concatenation of strings:

The **concatenation** of two strings is the string formed by writing the first string followed by the second string with no space.

(or)

Let **u**, **v** be two strings. **Concatenation** of strings **u** and **v** is appending symbols of **v** to right end of **u** i.e., **uv**

String Operations

i) Concatenation of strings:

Examples:

Ex-1:

$w_1 = DOG, w_2 = HOUSE$

$w_1w_2 = DOGHOUSE$

Ex-2:

$w_1 = x_1x_2\ldots x_n, w_2 = y_1y_2\ldots y_n$

$w_1w_2 = x_1x_2\ldots x_ny_1y_2\ldots y_n$

String Operations

ii) Length of a string:

The **length** of a string **w**, denoted by **|w|** is the number of symbols composing the string.

Examples:

Ex-1:

w_1 = abcd

$|w_1|$ = 4

Ex-2:

w_2 = aabbcc

$|w_2|$ = 6

String Operations

iii) Empty string:

The **empty** string is denoted by **λ** (lambda)or **ε** (epsilon)

is the string consisting of zero symbols (i.e., no symbols).

i.e., **|λ|** or **|ε| = 0**

Note:

εw = wε = w for any string **'w'**

String Operations

iv) Reverse of a string:

Let a string $w = a_1a_2\ldots a_n$ then the reverse of string w is denoted by w^R where R is reverse.

Therefore $w^R = a_na_{n-1}\ldots a_2a_1$

Example:

$w = abcd$

$w^R = dcba$

String Operations

v) Substring:

Let a string **w**. Any string of consecutive characters from **w** is called as **substring** of **w**.

Example:

w = xyz then **xy**, **yz** etc., are the substrings of **w**.

String Operations

v) Substring:

Prefix: A **`prefix`** of a string is any number of leading symbols (starting) of that string.

Suffix: A **`suffix`** of a string is any number of trailing symbols (last) of that string.

Proper prefix or proper suffix: A **`prefix`** or **`suffix`** of a string other than the string itself is called a **`proper prefix`** or a **`proper suffix`**.

String Operations

v) Substring:

Examples:

Let $w = xyz$

Prefixes: ε, x, xy, xyz

Suffixes: ε, z, yz, xyz

Proper prefixes: ε, x, xy

Proper suffixes: ε, z, yz

String Operations

vi) Sets of Strings:

If Σ is an alphabet then Σ^* (read it as sigma closure) is the set of strings obtained by concatenating 0 (zero) or more symbols from Σ .

Σ^* always contains ε

$\Sigma^+ = \Sigma^* - \{\varepsilon\}$ (Σ^+ read it as sigma positive closure)

Note: Σ is finite but Σ^* and Σ^+ are infinite.

String Operations

vi) Sets of Strings:

Example:

$\Sigma = \{a, b\}$

$\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, bab, bbb, \ldots\}$

$\Sigma^+ = \{a, b, aa, ab, ba, bb, aaa, aab, aba, bab, bbb, \ldots\}$

Introduction to Automata Theory:

For Lecture-01 watch the video lecture using the following link :

https://youtu.be/ZIp44Fn79Ak

Language

A **language** is a set of strings of symbols from some one alphabet (**Σ**).

A **language L** is subset of **Σ***

i.e., $L \subseteq \Sigma^*$

An empty set **ϕ** and the set consisting of empty string i.e., **{ε}** are **languages** and these two are distinct.

Language

Examples:

Ex-1.

The set of palindromes over the alphabet $\Sigma = \{0, 1\}$

is an infinite language.

The languages are

$\{\varepsilon, 0, 1, 11, 010, 101, 00100, \ldots\}$

Language

Examples:

Ex-2.

The set of all strings over a fixed alphabet '**a**', we denote this language by Σ^*

For example

i) If $\Sigma = \{a\}$ then

$\Sigma^* = \{\varepsilon, a, aa, aaa, aaaa, \ldots\}$

Language

Examples:

Ex-2. cont'd.

ii) If $\Sigma = \{a, b\}$ then

$\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aba, aab, bba, \ldots\}$

Language

Examples:

Ex-3.

$L = \{a^n b^n \mid n>0\}$ is an infinite language.

If $n = 1$, then $L = \{ab\}$

If $n = 2$, then $L = \{a^2b^2\} = \{aabb\}$

If $n = 3$, then $L = \{a^3b^3\} = \{aaabbb\}$

....

Language Operations

i) Compliment of a language:

The **compliment** of a language **L** is denoted by $\bar{L}$ or L' and is defined as

$$L' = \sum^* - L$$

Language Operations

ii) Reverse of a language:

The **reverse** of a language **L** is denoted by L^R and is defined as

$$L^R = \{w^R \mid w \in L\} \quad \text{for string } \mathbf{w}$$

Example:

$$L = \{xy \mid x, y \in w\}$$

$$L^R = \{yx \mid x, y \in w\}$$

Language Operations

iii) Concatenation of two languages:

Let two languages L_1 and L_2. The concatenation of two languages L_1 and L_2 is denoted by L_1L_2 and is defined as combining of the strings in both languages.

Example:

$$L_1 = \{x \mid x \in (a,b)^*\}$$

$$L_2 = \{y \mid y \in (a,b)^*\}$$

$$L_1L_2 = \{xy \mid x \in L_1, y \in L_2\}$$

Language Operations

iv) Closure operations of a language:

Given an **alphabet** (**Σ**), if we wish to define a language in which any strings of symbols from an **alphabet** (**Σ**) is a string including a null string is called as **star closure** or **kleen closure**.

$$\therefore L^* \approx \sum^*$$

Language Operations

iv) Closure operations of a language:

Given an **alphabet** (**Σ**), if we wish to define a language in which any strings of symbols from an **alphabet** (**Σ**) is a string excluding a null string is called as **positive** or **+ve closure**.

$$\therefore L^{+} \approx \sum^{+}$$

Language Operations

iv) Closure operations of a language:

$$L \subseteq \sum^*$$

$$L^* = \sum^*$$

$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots \cup L^n$$

where $L^0 = \{\varepsilon$ or $\lambda\}$

$$L^+ = L^* - L^0$$

$$\therefore L^+ = L^1 \cup L^2 \cup \ldots \cup L^n$$

$$\therefore L^+ = \sum^+$$

Language Operations

iv) Closure operations of a language:

similarly

$$\sum^* = \sum^0 \cup \sum^1 \cup \ldots \cup \sum^n$$

i.e., Σ^* is the set of all strings on Σ can be interpreted as star closure.

if Σ^1 is interpreted as the language consisting of all strings of length 1. With this interpretation we may write :

$$\sum^* = \sum^0 \cup \sum^1 \cup \ldots \cup \sum^n$$

Language Operations

iv) Closure operations of a language:

For example:

Σ^2 is the set of all strings of length ‘2’ over Σ.

Σ^3 is the set of all strings of length ‘3’ over Σ.

…..

Σ^k is the set of all strings of length ‘k’ over Σ.

In general

$\Sigma^* = \{\{\varepsilon\} \cup \{\text{strings with length } 1\} \cup \{\text{strings with length } 2\} \cup \ldots \cup \{\text{strings with length } k\} \ldots\}$

Language Operations

iv) Closure operations of a language:

Example:

$\Sigma = \{a, b\}$

$\Sigma^1 = \{a, b\} = \Sigma$

$\Sigma^2 = \{aa, ab, ba, bb\}$

$\Sigma^* = \{\{\varepsilon\} \cup \{a,b\} \cup \{aa,ab, ba, bb\} \cup \ldots\}$

Formal Language

A set of strings of symbols from some one alphabet (**Σ**) is called a **formal language**.

The set may be:

- Empty set
- Finite set
- Infinite set

Example:

L = {ε, a, aa, aaa, …}

L(M) is a language defined or accepted or recognized by a machine or abstract model or recognizer **M**.

Grammar or Formal Grammar

A **Grammar** or **Formal Grammar** describes the structure of a language.

A **Grammar** is denoted by **G** and is defined as a **4-tuple** i.e., **G = (V, T, S, P)**

Where

V is non empty set of symbols called as **Variables**

T is non empty set of symbols called as **Terminals**

S∈V is a **Start Variable**

P is set of **productions** or **production rules**

Grammar or Formal Grammar

General form of Productions:

$$P : \alpha \rightarrow \beta$$

$$\alpha \in (V \cup T)^{+}$$

$$\beta \in (V \cup T)^{*}$$

Grammar or Formal Grammar

Notations:

- **`Variables`** are denoted by only UPPER CASE letters and some special Greek letters etc. Variables are also called a Non-Terminals.
- **`Terminals`** are denoted by lower case letters i.e., **`a to z`** and digits **`0 to 9`** and some special operators like arithmetic operators, relational operators etc.

Grammar or Formal Grammar

Example-1:

Let **G = (V, T, S, P)** is a grammar.

Where

V = {S, A, B}

T = {a, b}

S is a Start Variable

And Productions **P** are given below:

$$S \rightarrow ASB$$

$$A \rightarrow aSb \mid \varepsilon$$

$$B \rightarrow bSa \mid \varepsilon$$

ε or **λ** is Null String

Grammar or Formal Grammar

Example-2:

Let **G = (V, T, S, P)** is a grammar.

Where

V = {S, A, B}

T = {a, b, +}

S is a Start Variable

And Productions **P** are given below:

$$Sa \rightarrow ASB$$

$$Sb \rightarrow aSb \mid \varepsilon$$

$$BA \rightarrow bSB \mid A + B$$

ε or **λ** is Null String

CHOMSKY HIERARCHY OF FORMAL LANGUAGES

Chomsky Hierarchy of Formal Languages :

The famous linguistic **Noam Chomsky** attempted to formalize the notion of grammar and languages in the 1950s. This effort, due to **Chomsky**, resulted in the definition of the "**Chomsky Hierarchy**", a hierarchy of language classes defined by gradually increasing the restrictions on the form of the productions. **Chomsky** numbered the four families of grammars (and languages) that make up the hierarchy and are defined as below.

Chomsky Hierarchy of Formal Languages :

Type	Languages	Grammars	Automata
0	Recursively Enumerable Language (REL)	Phrase-structured/ Semi-true/Unrestricted grammars	Turing Machine (TM)
1	Context-Sensitive Language (CSL)	Context-Sensitive Grammars (CSG)	Linear-Bounded Automata (LBA)
2	Context-Free Language (CFL)	Context-Free Grammars (CFG)	Pushdown Automata (PDA)
3	Regular Language (RL)	Regular, Right-linear, Left-linear grammar	Finite Automata (FA)

Chomsky Hierarchy of Formal Languages :

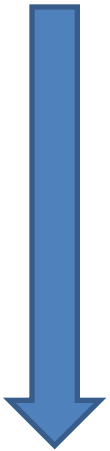
Computational power in decreasing order:

Type 0

Type 1

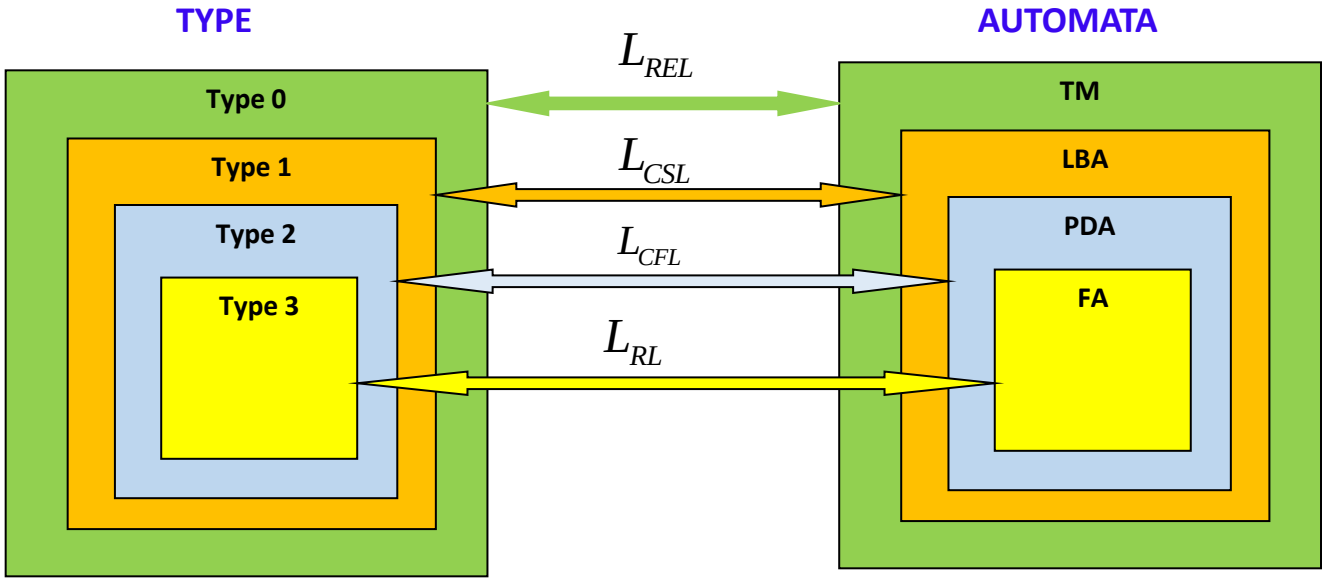
Type 2

Type 3



Type-3 < Type-2 < Type-1 < Type-0

Chomsky Hierarchy of Formal Languages :



Chomsky Hierarchy of Formal Languages:

For full details on Chomsky Hierarchy of Languages, watch the video lecture link given below :

https://youtu.be/RRKqqOmSLe8

Fundamental Concepts of Automata Theory:

For Lecture-02 watch the video lecture using the following link :

https://youtu.be/_ds04m7YjwU

Summary of Lecture- 01

- Introduction to Automata Theory or Theory of Computation
- Applications of Automata Theory or Theory of Computation
- Basic Fundamentals /Central Concepts of Automata Theory
- Formal Languages and Chomsky Hierarchy