# Last Minute Notes – Theory of Computation

Last Updated : 24 Jan, 2025

The Theory of Computation (TOC) is a critical subject in the GATE Computer Science syllabus. It involves concepts like Finite Automata, Regular Expressions, Context-Free Grammars, and Turing Machines, which form the foundation of understanding computational problems and algorithms.

This article provides Last Minute Notes for TOC, focusing on the most important topics that are frequently asked in GATE.

**Table of Content**

## Basics

### 1. Symbol and Alphabet

- **Symbol**: A single character or entity, e.g., `a, b, 1, 0`.
- **Alphabet (Σ)**: A finite set of symbols, e.g., `Σ = {a, b}`.

### 2. String

- **String**: A finite sequence of symbols from an alphabet, e.g., `abba`.
- **Empty String (ε)**: A string with no symbols.

### 3. Operations on Strings

- **Concatenation**: Joining two strings.
  Example: `w1 = ab`, `w2 = ba` ⟶ `w1.w2 = abba`.
- **Length (|w|)**: Number of symbols in a string.
  Example: `|abba| = 4`.

- **Reverse (w^R)**: Reversing the order of symbols.
  Example: `w = abba` ⟶ `w^R = abba`.

#### 4. Prefix, Suffix, and Substring

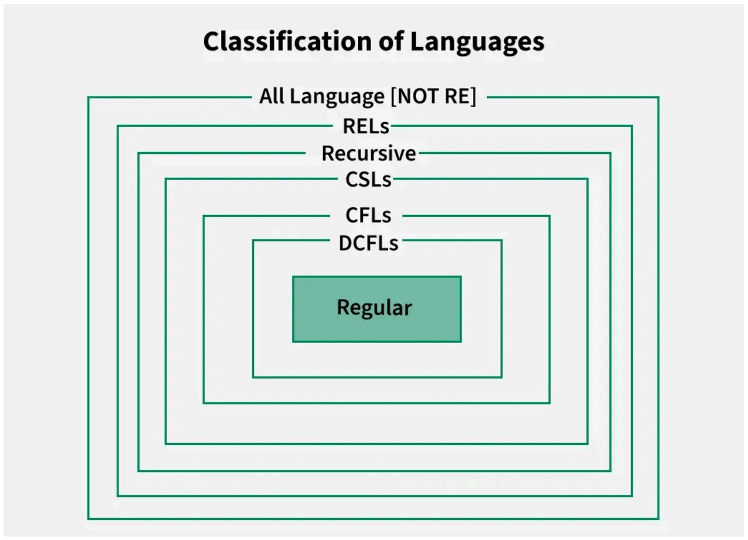
- **Prefix**: Any leading part of a string.
  Example: For `w = abba`, prefixes are `{ε, a, ab, abb, abba}`.
- **Suffix**: Any trailing part of a string.
  Example: For `w = abba`, suffixes are `{ε, a, ba, bba, abba}`.
- **Substring**: Any continuous part of a string.
  Example: For `w = abba`, substrings are `{ε, a, b, ab, ba, bb, abb, bba, abba}`.

#### 5. Language

- **Language (L)**: A set of strings over an alphabet.
  Example: `L = {w ∈ Σ* | w starts with a and ends with b}`.
  For `Σ = {a, b}`, `L = {ab, aab, abb, ...}`.

### Types of Languages

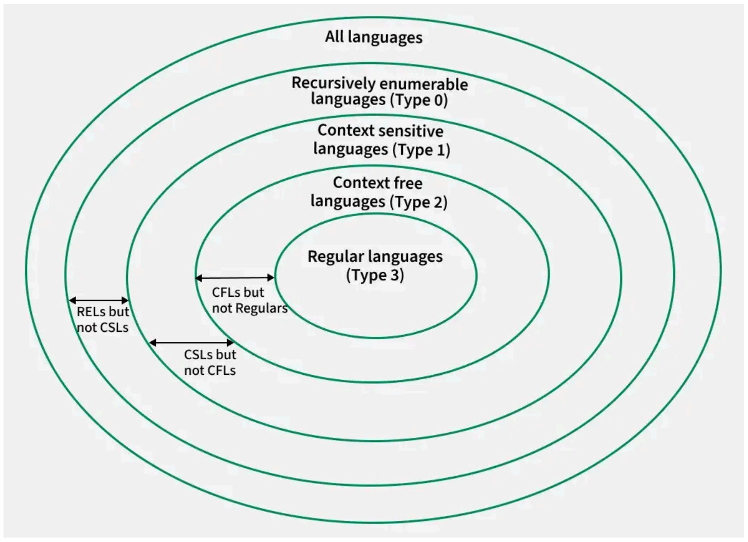
• Finite Language • Infinite Language • Regular language • DCFL (Deterministic CFL) • CFL • CSL • Recursive Language • Recursive Enumerable Language (REL)



### Relation Between Symbol, Alphabet, String, and Language

1. **Symbol**: The smallest unit, e.g., `a`, `b`, `0`, `1`.
2. **Alphabet (Σ)**: A finite set of symbols, e.g., `Σ = {a, b}`.
3. **String**: A sequence of symbols from an alphabet, e.g., `abba`.
4. **Language**: A set of strings over an alphabet, e.g., `L = {ab, aab, abb}`.

### Chomsky Hierarchy


All languages
Recursively enumerable languages (Type 0)
Context sensitive languages (Type 1)
Context free languages (Type 2)
Regular languages (Type 3)
CFLs but not Regulars
RELs but not CSLs
CSLs but not CFLs

*Chomsky Hierarchy*

Read more about [Chomsky Hierarchy in TOC.]()

## Finite Automata

Finite Automata (FA) is a simple mathematical model used to represent and recognize regular languages. FA = (Q, Σ, δ, $q_0$ , F)

Finite Automaton can be categorized into two types:

Acceptor (Without Output)

- DFA (Deterministic Finite Automaton)
- NFA (Non-Deterministic Finite Automaton)

Transducer (With Output)

- Moore Machine
- Mealy Machine

### Types of FA:

- **Deterministic Finite Automata (DFA)**: Transition function: δ: Q × Σ → Q (Maps a state and input symbol to a single next state).
- **Non-Deterministic Finite Automata (NFA)**:
  - Without ε-moves: δ: Q × Σ → 2^Q
    (Maps a state and input symbol to a set of possible states).
  - With ε-moves: δ: Q × (Σ ∪ {ε}) → 2^Q
    (Maps a state and input symbol or ε to a set of possible states).

Read more about Introduction to FA.

**Note:**

- *Language accepted by NDFA and DFA are same.*
- *Power of NDFA and DFA is same.*
- *No. of states in NDFA is less than or equal to no. of states in equivalent DFA.*
- *For NFA with n-states, in worst case, the maximum states possible in DFA is* $2^n$
- *Every NFA can be converted to corresponding DFA.*

## Steps to Construct a DFA:

- Identify the input alphabet (Σ) and the states (Q).
- Define the initial state and the final states.
- Create a transition table or diagram ensuring every input symbol from each state leads to exactly one state.
- Ensure the DFA accepts all strings of the given language and rejects others.

## Steps to Construct an NFA:

- Identify the pattern or condition the NFA should accept.
- Create states for each stage of processing the input.
- Start from an initial state.
- Define one or more final states based on acceptance criteria.
- Allow multiple transitions for the same input symbol.
- Include ε-transitions if needed (moves without consuming input).
- Check if it accepts all strings in the language and rejects others.

Simple and flexible, NFAs are easier to design than DFAs.

## NFA to DFA Conversion

Step 1: Convert the given NFA to its equivalent transition table.
Step 2: Create the DFA's start state.
Step 3: Create the DFA's transition table.
Step 4: Create the DFA's final states.

Step 5: Simplify the DFA.
Step 6: Repeat steps 3-5 until no further simplification is possible.

Read more about NFA to DFA Conversion, Here.

## Minimization of DFA

Suppose there is a DFA D < Q, Δ, q0, Δ, F > which recognizes a language L. Then the minimized DFA D < Q', Δ, q0, Δ, F' > can be constructed for language L as:
**Step 1:** We will divide Q (set of states) into two sets. One set will contain all final states and other set will contain non-final states. This partition is called $P_0$.
**Step 2:** Initialize k = 1
**Step 3:** Find $P_k$ by partitioning the different sets of $P_{k-1}$. In each set of $P_{k-1}$, we will take all possible pair of states. If two states of a set are distinguishable, we will split the sets into different sets in $P_k$.
**Step 4:** Stop when $P_k = P_{k-1}$ (No change in partition)
**Step 5:** All states of one set are merged into one. No. of states in minimized DFA will be equal to no. of sets in $P_k$.
**How to find whether two states in partition $P_k$ are distinguishable ?**
Two states ( qi, qj ) are distinguishable in partition $P_k$ if for any input symbol a, Δ ( qi, a ) and Δ ( qj, a ) are in different sets in partition $P_{k-1}$.

Read more about Minimization of DFA.

## Moore and Mealy Machine

### Moore Machine:

- Output depends on the state only.
- Output is produced when entering a state.
- Example: Output = `{q0 → 0, q1 → 1}`.
- Representation: M=(Q,Σ,Δ,δ,λ,q0) Where λ is the output function.

### Mealy Machine:

- Output depends on the state and input.
- Output is produced during transitions.
- Example: Transition `q0 → q1` on `a` produces `1`.
- Representation: M=(Q,Σ,Δ,δ,λ,q0) Where λ is the transition-based output function.

**Note**: Mealy Machines often require fewer states than Moore Machines for the same functionality.

Read about Mealy and Moore Machine, Here.

### Regular Expression

A regular expression represents a regular language and describes a regular set.

#### Operators of Regular Expressions

**OR (|)**: Binary Operator, Combines two patterns, Example: `a|b` → `{a, b}`.

**Concatenation (.):** Binary Operator, Joins two patterns in sequence, Example: `ab` → `{ab}`.

**Kleene Star (*)**: Unary Operator, Allows repetition (zero or more times), Example: `a*` → `{ε, a, aa, aaa, ...}`.

**Kleene Plus (+)**: Unary Operator, Allows repetition (one or more times), Example: `a+` → `{a, aa, aaa, ...}`.

**Language Over Σ**

Languages over an alphabet (Σ) can be classified as:

**Finite Set**:

- Always a Regular Language.

**Infinite Set**:

- L Over |Σ| = 1:
    - If it forms an Arithmetic Progression (AP) → Regular Language.
    - If it does not form an AP → Non-Regular.
- L Over |Σ| > 1:
    - Can be either a Regular Language or a Non-Regular Language, depending on the conditions.

### Identification of Regular Languages

**1. Finite Languages are Always Regular**

**2. Infinite Languages are Regular if they Follow Patterns**

- Infinite languages are regular if they can be expressed with a finite automaton or a regular expression.
- Example:
  - `L = {a^n | n ≥ 0}` (Strings with any number of `a`s) → Regular, as it can be represented by a DFA with loops.
  - `L = {a^n b^m | n, m ≥ 0}` → Regular because it doesn't require memory to match `n` with `m`.

#### 3. Closure Properties of Regular Languages

- Regular languages are not closed under subset operation and the six infinite operations because these operations often require memory, context, or infinite processing, which finite automata lack.
- Finite operations, like union, concatenation, and intersection (over finite languages), are closed because they can be managed within the scope of finite automata.

Read about Closure Properties of Regular Languages, Here.

#### 4. How to Identify Non-Regular Languages

A language is not regular if:

1. **Memory is required**: The language needs to keep track of counts or comparisons. Example: `L = {a^n b^n | n ≥ 0}` → Non-regular, as it requires matching the number of `a`s and `b`s.
2. **Nested Patterns**: The language contains self-referencing structures like palindromes.
   Example: `L = {ww^R | w ∈ Σ*}` → Non-regular.

#### 5. Pumping Lemma Test

- **Purpose**: To prove that a language is not regular.
- Statement: For any regular language `L`, there exists a pumping length `p` such that any string `w` in `L` with `|w| ≥ p` can be split into `xyz` where:
  1. `xy^iz ∈ L` for all `i ≥ 0`.
  2. `|y| > 0`.
  3. `|xy| ≤ p`.

Read about Properties of RegEx, Here.

## Arden’s Theorem

**Statement**:

- Arden’s Theorem provides a method to solve regular expression equations of the form:
  R=Q+RP, where R,Q,P are regular expressions.

**Solution**:

- The solution for R is:
  R=QP^*if P does not contain ε (epsilon).

**Example**:
Given R=a+Rb:

- Using Arden’s Theorem: R = a(b^*).

Read about Arden’s Theorem, Here.

## Regular Grammar

**Definition**: A regular grammar is a formal grammar that generates regular languages.

**Types of Regular Grammar**:

- **Right-Linear Grammar**:
  Productions are of the form:
  A→aB or A→a, where A,B are non-terminals and a is a terminal.
- **Left-Linear Grammar**:
  Productions are of the form:
  A→Ba or A→a.

**Properties**:

- Regular grammars correspond to finite automata.
- Right-linear grammars are used for DFA/NFA construction.

**Example** :

**Grammar** :

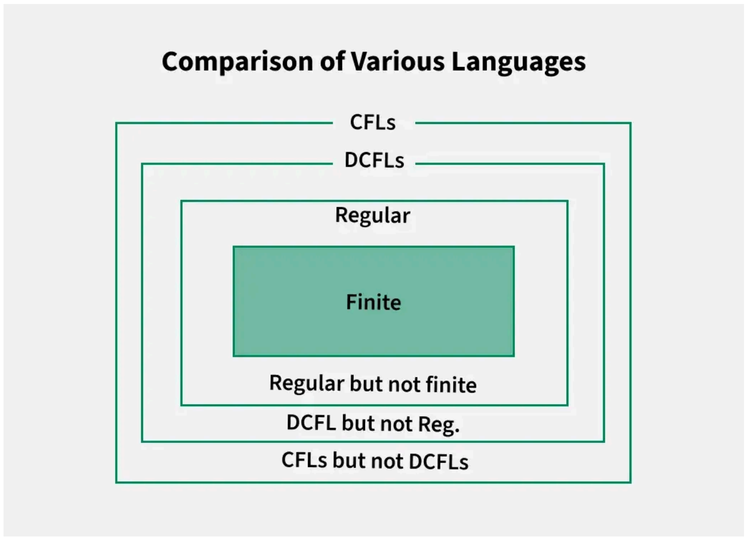
- S→Aa|Ba
- A→a
- B→b

**Language (L)**: L={aa,bb}

- Regular languages generated from grammars are calculated bottom-to-top (from the base rules upward).
- Use substitution to derive the final language.

Read about Regular Grammar, Here.

## Push Down Automata



### Context Free Grammar

**Definition**: A context-free language is generated by a Context-Free Grammar (CFG) where each production rule follows:
V → (V ∪ T)*

- **LHS** (Left-Hand Side): Only one variable (non-terminal).
- **RHS** (Right-Hand Side): A combination of terminals and/or non-terminals.

**Derivations to Generate Strings**:

- **Linear Derivation**:
  (a) **Left Most Derivation (LMD)**: Expand the leftmost variable first.
  (b) **Right Most Derivation (RMD)**: Expand the rightmost variable first.
- **Non-Linear Derivation**:
    - Also called Parse Tree or Derivation Tree.

  - Represents a hierarchical structure of derivation.

### Types of Context-Free Grammars (CFG)

**Ambiguous Grammar**:

- A CFG is ambiguous if a string has more than one parse tree or derivation.
- Example: $S \to SS \mid a$.

**Unambiguous Grammar**:

- A CFG is unambiguous if every string has exactly one parse tree or derivation.

**Left-Recursive Grammar**:

- A grammar is left-recursive if a production rule has the form $A \to A\alpha$, where $\alpha$ is a string.
- Example: $S \to Sa \mid b$.

**Right-Recursive Grammar**:

- A grammar is right-recursive if a production rule has the form $A \to \alpha A$, where $\alpha$ is a string.
- Example: $S \to aS \mid b$.

**Regular Grammar**:

- A special type of CFG where rules are either right-linear or left-linear.

Read about Context Free Grammar, Here.

## PDA

A PDA is a computational model that extends a finite automaton by using a stack for additional memory. It recognizes context-free languages (CFLs).

Read about Introduction to Pushdown Automata, Here.

#### Types of Pushdown Automata (PDA)

Pushdown Automata (PDA) are used to recognize context-free languages (CFLs), classified into deterministic context-free languages (DCFLs) and general CFLs.

##### 1. Deterministic Pushdown Automata (DPDA)

- **Definition**: A DPDA allows at most one transition for each combination of input symbol, current state, and stack symbol.
- **Recognizes**: Deterministic Context-Free Languages (DCFLs), which are a subset of CFLs.
- **Characteristics**:
    - Cannot handle **ambiguity** (e.g., ambiguous grammars).
    - Requires deterministic parsing.
    - Example language: L =$\{a^n b^n | n \geq 0\}$(balanced strings).
- **Usage**:
    - Recognizes languages with clear, unambiguous structures.

#### 2. Non-Deterministic Pushdown Automata (NPDA)

- **Definition**: An NPDA allows multiple transitions for the same input symbol, current state, and stack symbol.
- **Recognizes**: All Context-Free Languages (CFLs).
- **Characteristics**:
    - Can handle ambiguity (e.g., ambiguous grammars).
    - Example language: L = $\{a^n b^m c^n | n, m \geq 0\}$.
    - Supports more complex structures than DPDAs.
- **Usage**:
    - Recognizes all CFLs, including ambiguous languages.

**Note:**

- Power of NPDA is more than DPDA.
- It is not possible to convert every NPDA to corresponding DPDA.
- Language accepted by DPDA is subset of language accepted by NPDA.
- The languages accepted by DPDA are called DCFL (Deterministic Context Free Languages) which are subset of NCFL (Non Deterministic CFL) accepted by NPDA.

### Key Difference Between DPDA and NPDA

| Feature | DPDA | NPDA |
|---|---|---|
| **Language Recognized** | DCFL (Subset of CFL) | CFL (All Context-Free Languages) |

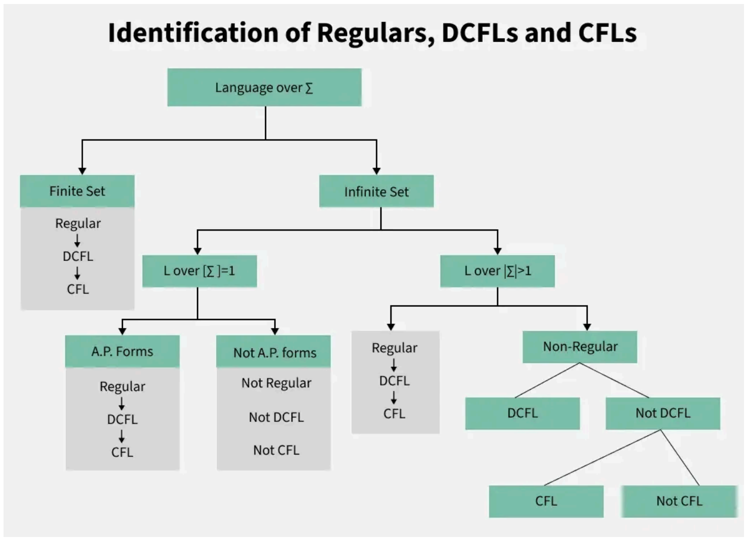
| Feature | DPDA | NPDA |
|---|---|---|
| **Transitions** | Single transition per input | Multiple transitions allowed |
| **Ambiguity** | Cannot handle ambiguous grammars | Can handle ambiguous grammars |

Read about Difference Between DPDA and NPDA, Here.

### Closure Properties of CFLs and DCFLs

| Operation | CFL | DCFL |
|---|---|---|
| **Union ($L1 \cup L2$)** | **Closed** | **Not Closed** |
| **Concatenation ($L1 \cdot L2$)** | **Closed** | **Not Closed** |
| **Kleene Star ($L^*$)** | **Closed** | **Not Closed** |
| **Intersection ($L1 \cap L2$)** | **Not Closed** | **Closed with Regular** |
| **Complement ($L^c$)** | **Not Closed** | **Closed** |
| **Reversal ($L^R$)** | **Closed** | **Not Closed** |
| **Homomorphism** | **Closed** | **Not Closed** |
| **Substitution** | **Closed** | **Not Closed** |
| **Intersection with Regular** | **Closed** | **Closed** |
| **Difference ($L1 – L2$)** | **Not Closed** | **Not Closed** |

Read about Closure Properties of CFLs, Here.



Key Points:

- DCFL ∪ Regular = DCFL
- DCFL ∩ Regular = DCFL
- DCFL – Regular = DCFL
- Regular – DCFL = Regular
- DCFL ∪ CFL = CFL (Need not be DCFL)
- DCFL ∩ CFL = Need not be CFL
- DCFL – CFL = Need not be CFL
- CFL – DCFL = Need not be CFL
- DCFL ∪ Finite = DCFL
- DCFL ∩ Finite = Finite
- DCFL – Finite = DCFL
- Finite – DCFL = Finite

## Turing Machine

- **Definition**:

  A Turing Machine (TM) is a mathematical model of computation used to define what can be computed.

**Components**:

- **Q**: Finite set of states.
- **Σ**: Input alphabet (does not include the blank symbol).
- **Γ**: Tape alphabet (Σ⊆Γ, includes the blank symbol).
- **δ**: Transition function.
- **$q_0$**: Initial state ($q_0$∈Q\).
- **q_accept**: Accepting state.

- **q_reject**: Rejecting state ($q\_accept \neq q\_reject$).

**Transition Function**:

- **DTM**: $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L,R\}$
- **NTM**: $\delta: Q \times \Gamma \to 2Q \times \Gamma \times \{L,R\}$

**Types of Turing Machines**:

- **DTM (Deterministic TM)**: One transition for each state-symbol pair.
- **NTM (Non-Deterministic TM)**: Multiple transitions for each state-symbol pair.

**Language Classification**:

- **Turing Recognizable (Recursively Enumerable)**:
  Languages accepted by a TM (TM halts for strings in the language but may loop for others).
- **Turing Decidable (Recursive)**:
  Languages for which the TM halts on every input.

**Special Types of TMs**:

- **Multi-Tape TM**: Multiple tapes and heads; equivalent in power to a single-tape TM.
- **Multi-Track TM**: Single tape divided into multiple tracks.
- **Non-Deterministic TM**: Simulates multiple computation paths; equivalent in power to DTM.
- **Universal TM**: Simulates any TM by encoding its description.

**Key Properties**:

- TM can simulate Finite Automata and Pushdown Automata.
- TM is more powerful than DFA, NFA, and PDA.
- TM accepts Recursively Enumerable Languages.

Read more about Turing Machine in TOC, Here.

**Church-Turing Thesis**:

- Any computation performed by a mechanical process can be simulated by a Turing Machine.

Read about Chruch-Turing Thesis, Here.

**Important Points**:

- TMs are used to define Decidability and Undecidability.
- Halting Problem: Classic example of an undecidable problem.
- Equivalence of DTM and NTM: Both recognize the same set of languages.
- Multi-tape TMs are computationally equivalent to single-tape TMs but more efficient.
- Language accepted by NTM, multi-tape TM and DTM are same.
- Power of NTM, Multi-Tape TM and DTM is same.
- Every NTM can be converted to corresponding DTM.

**Time Complexity**:

- Deterministic TM: O(f(n)) for time complexity f(n).
- Non-Deterministic TM: $O(2^{O(f(n))})$ in the worst case when converted to DTM.

## Recursive and Recursive Enumerable Language

**Recursive Language (Decidable Language)**: A language is recursive if there exists a Turing Machine (TM) that halts on every input and correctly decides whether the input is in the language.

**Recognizable or Recursively Enumerable Language (Semi-Decidable Language):** A language is recursively enumerable if there exists a Turing Machine (TM) that halts and accepts inputs that belong to the language, but it may loop forever for inputs not in the language.

Read about Recursive and Recursive Enumerable Language in TOC, Here.

**Complement Property of Rec and RE Language:**
Complement of Recursive set is Recursive.
Complement of RE is either Recursive or non-RE.
Complement of RE never be “RE which is not recursive”.

**Closure Properties of RE and Rec Language**

| Operation | Recursive Languages (Rec) | Recursively Enumerable Languages (RE) |
|---|---|---|
| **Union** | Closed | Closed |

| Operation | Recursive Languages (Rec) | Recursively Enumerable Languages (RE) |
|---|---|---|
| **Intersection** | Closed | Not Closed |
| **Complement** | Closed | Not Closed |
| **Concatenation** | Closed | Closed |
| **Kleene Star** | Closed | Closed |
| **Difference** | Closed | Not Closed |
| **Reversal** | Closed | Closed |
| **Intersection with Regular** | Closed | Closed |

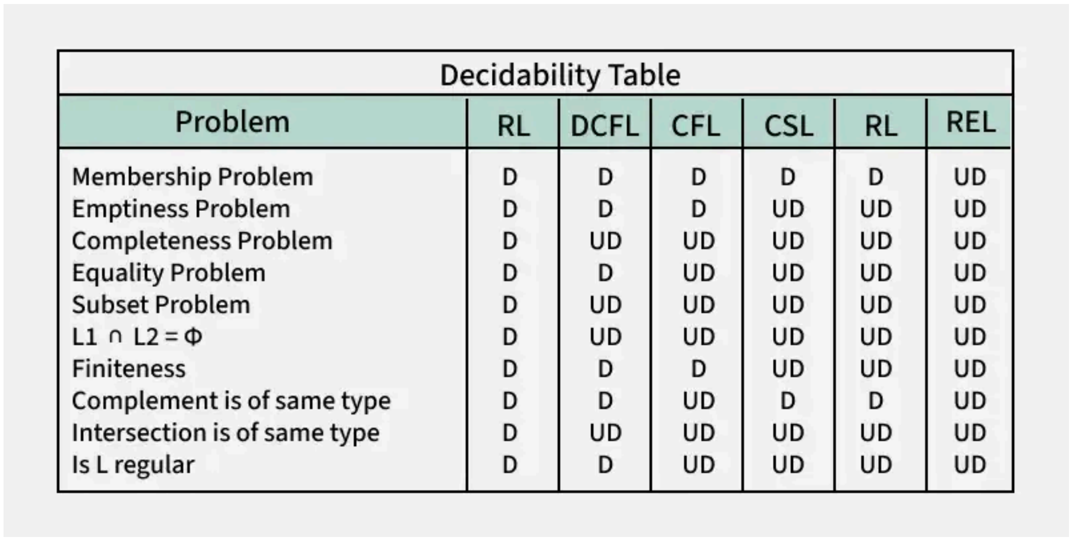
**Decidable Problems (Recursive Languages)**:

- A problem is decidable if there exists a Turing Machine (TM) that halts for all inputs (accepts for “yes” and rejects for “no”).
- Examples include membership and equivalence problems for regular and context-free languages.

**Undecidable Problems**:

- Problems for which no Turing Machine (HTM) exists to solve them for all inputs.
- Further classified into:
    - **Recursively Enumerable (RE) but not Recursive**:
        - The problem is semi-decidable (a TM exists that halts for “yes” cases but may loop forever for “no”).
    - **Not Recursively Enumerable (Not RE)**:
        - No Turing Machine exists to even semi-decide the problem.

Read about Decidable and Undecidable Problem in TOC, Here.



| Decidability Table | | | | | | |
|---|---|---|---|---|---|---|
| Problem | RL | DCFL | CFL | CSL | RL | REL |
| Membership Problem | D | D | D | D | D | UD |
| Emptiness Problem | D | D | D | UD | UD | UD |
| Completeness Problem | D | UD | UD | UD | UD | UD |
| Equality Problem | D | D | UD | UD | UD | UD |
| Subset Problem | D | UD | UD | UD | UD | UD |
| L1 ∩ L2 = Φ | D | UD | UD | UD | UD | UD |
| Finiteness | D | D | D | UD | UD | UD |
| Complement is of same type | D | D | UD | D | D | UD |
| Intersection is of same type | D | UD | UD | UD | UD | UD |
| Is L regular | D | D | UD | UD | UD | UD |

### Countability in Turing Machines

1. **Countable Set of Turing Machines**:
   - The set of all possible **Turing Machines** is countable because each Turing Machine can be encoded as a finite string (its description can be encoded using a finite alphabet, making it possible to enumerate all possible TMs).

2. **Uncountable Set of Languages**:
   - The set of **all possible languages** (subsets of all possible strings) is uncountable, as there are more possible languages than there are Turing Machines. This is due to the fact that we can define languages using infinite sets, and there are more subsets of an infinite set than the number of elements in the set.

Read more about Determining Countability in TOC, Here.

See Last Minute Notes on all subjects , here.

Comment More info Advertise with us

**Next Article**

Theory of Computation - GATE CSE Previous Year Questions

## Similar Reads

**Introduction to Computation Complex Theory**

Broad Overview : Complexity theory, in a nutshell, a complexity word is a quite fancy word, literally, it sounds complex, but it is not an intimidating topic. What it really...

4 min read

**Chomsky Hierarchy in Theory of Computation**

According to Chomsky hierarchy, grammar is divided into 4 types as follows: Type 0 is known as unrestricted grammar.Type 1 is known as context-sensitive grammar.Type 2...

2 min read

**Pumping Lemma in Theory of Computation**

There are two Pumping Lemmas, which are defined for 1. Regular Languages, and 2. Context - Free Languages Pumping Lemma for Regular Languages For any regular...

4 min read

**Theory of Computation | Regular languages and finite automata | Question 2**

What is the complement of the language accepted by the NFA shown below? (A) A (B) B (C) C (D) D Answer: (B) Explanation: Quiz of this QuestionPlease comment below if...

1 min read

**Relationship between grammar and language in Theory of Computation**

In the Theory of Computation, grammar and language are fundamental concepts used to define and describe computational problems. A grammar is a set of production rule...

4 min read

**Arden's Theorem in Theory of Computation**

Arden's Theorem is a fundamental result in the Theory of Computation used to solve regular expressions from a given linear equation. It is particularly useful when...

6 min read

### Halting Problem in Theory of Computation

The halting problem is a fundamental issue in theory and computation. The problem is to determine whether a computer program will halt or run forever.Definition: The...

4 min read

### Decidability Table in Theory of Computation

Prerequisite - Undecidability, Decidable and undecidable problems Identifying languages (or problems*) as decidable, undecidable or partially decidable is a very...

2 min read

### Introduction To Grammar in Theory of Computation

In the context of the Theory of Computation, grammar refers to a formal system that defines how strings in a language are structured. It plays a crucial role in determining...

4 min read

### Decidable and Undecidable Problems in Theory of Computation

In the Theory of Computation, problems can be classified into decidable and undecidable categories based on whether they can be solved using an algorithm. A...

6 min read