PART – A

QUESTION BANK

Q.No.	Question
1	A Hasse diagram is used to represent_____ **A) Partially ordered sets** B) Graphs C) Functions D) Binary Trees
2	Infinite sets can be classified as_____ A) Rational and irrational B) Real and complex **C) Countable and uncountable** D) Open and closed
3	Which operation on sets is idempotent? **A) Intersection** B) Union C) Difference D) Complement
4	Function $f(x) = x^2$ is_____ **A) Many-to-one** B) One-to-one C) Onto D) Bijective
5	Which data structure is best modelled using trees? A) Stack **B) Hierarchical structure** C) Queue D) Hash table
6	What does a function relate? A. One input to multiple outputs B. Many inputs to one output **C. Each input to exactly one output** D. Many outputs to one input
7	What best describes a partial order? A. Symmetric and reflexive **B. Antisymmetric and transitive** C. Transitive and symmetric D. Reflexive and symmetric
8	In set theory, what does the union of two sets A and B represent? A. Elements common to both A and B B. Elements unique to A **C. Elements in A or B or both** D. Elements not in A
9	What does the composition of relations represent in database queries? A. Filtering of data **B. Joining of tables** C. Sorting of rows D. Projection of attributes
10	. Which relation property does NOT apply to "is ancestor of"? A. Transitive B. Antisymmetric **C. Reflexive** D. None of the above
11	Which of the following is a well-formed formula? A) P ∨ ∧ Q **B) P ∧ (Q ∨ R)** C) ∧P Q D) (→P Q)
12	Interpret the semantic meaning of the formula: ¬(P ∨ Q)_____ A) P is false B) Q is false **C) Both P and Q are false** D) Either P or Q is false
13	In CNF, why is disjunction used inside clauses? A) To confuse the SAT solver B) To ensure validity C) To match truth tables **D) To ensure conjunction of disjunctive components**
14	A formula is satisfiable if_____ **A) There is at least one interpretation that makes it true** B) It is true in all cases C) It is invalid D) It is always false
15	The CNF of (P → Q) is equivalent to_____ A) P ∧ Q B) Q ∧ ¬P C) ¬Q ∨ P **D) ¬P ∨ Q**
16	What is a well-formed formula (wff) in propositional logic? A. A grammatically incorrect expression

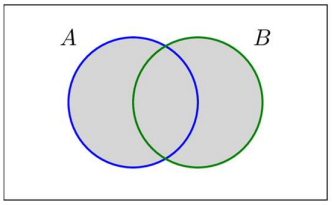
	B. An expression with undefined symbols **C. A syntactically correct formula based on rules** D. A formula with only conjunctions
17	In a truth table, how many rows are needed for a formula with 3 propositional variables? A. 6 B. 4 **C. 8** D. 2
18	Which of the following is the correct implication of satisfiability? A. A satisfiable formula must be valid B. A satisfiable formula is true in all interpretations **C. A satisfiable formula is true in at least one interpretation** D. A satisfiable formula is false in every interpretation
19	Which transformation helps simplify a formula for SAT solvers? A. First-order logic B. Converting to DNF **C. Converting to CNF** D. Resolution elimination
20	Which symbol represents logical equivalence? A. → B. ∨ **C. ↔** D. ∧

PART - B

Derive and explain different set operations with a Venn diagram.

1. Union (A ∪ B)

Definition: The union of two sets A and B is the set of all elements that are in A, in B, or in both.



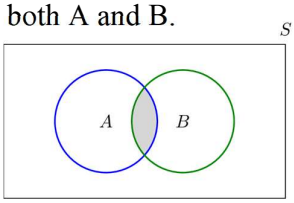
Symbolically:
$A \cup B = \{ x \mid x \in A \text{ or } x \in B \}$

Venn Diagram: All regions covered by A or B are shaded.

---

2. Intersection (A ∩ B)

Definition: The intersection of A and B is the set of all elements common to both A and B.



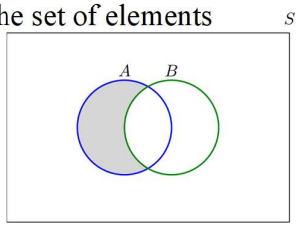
Symbolically:
$A \cap B = \{ x \mid x \in A \text{ and } x \in B \}$

Venn Diagram: Only the overlapping region between A and B is shaded.

---

3. Difference (A - B or A \ B)

Definition: The difference of A and B (also called relative complement) is the set of elements in A that are not in B.



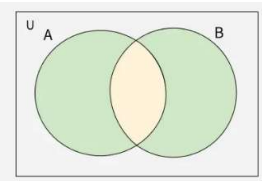
Symbolically:
$A - B = \{ x \mid x \in A \text{ and } x \notin B \}$

Venn Diagram: Only the part of A that does not intersect with B is shaded.

---

4. Symmetric Difference (A Δ B)

Definition: The symmetric difference of A and B is the set of elements which are in either of the sets, but not in their intersection.



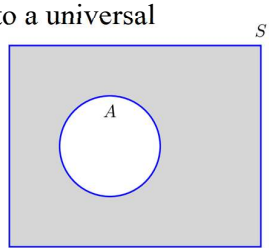
Symbolically:
$A \Delta B = (A - B) \cup (B - A)$

Venn Diagram: Regions unique to A and unique to B are shaded (excluding the intersection).

---

5. Complement (A^c)

Definition: The complement of a set A contains all elements not in A, relative to a universal set U.



Symbolically:
$A^c = \{ x \mid x \in U \text{ and } x \notin A \}$

Venn Diagram: The area outside A, but within the universal set U, is shaded.

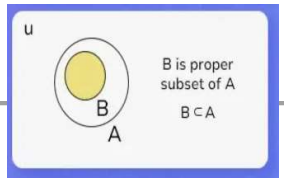
---

6. Subset ($A \subseteq B$)

Definition: Set A is a subset of B if every element of A is also an element of B.

Symbolically:
$A \subseteq B \Leftrightarrow \forall x (x \in A \rightarrow x \in B)$

Venn Diagram: Circle A lies entirely within circle B.



---

7. Proper Subset ($A \subset B$)

Definition: A is a proper subset of B if $A \subseteq B$ but $A \neq B$ (A has fewer elements).

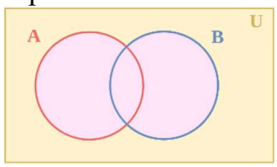
Symbolically:
$A \subset B \Leftrightarrow A \subseteq B \text{ and } \exists x (x \in B \text{ and } x \notin A)$

Venn Diagram: Same as subset, but with the additional condition that A doesn't cover all of B.

---

8. Universal Set (U)

Definition: The universal set contains all elements under consideration for a particular discussion.

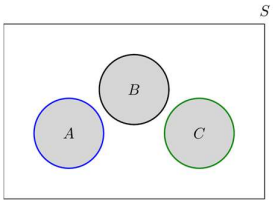


Symbolically:
$U = \{ \text{all possible elements} \}$

Venn Diagram: Usually shown as a rectangle enclosing all other sets.

---

9. Disjoint Sets



Definition: Two sets are disjoint if they have no elements in common.

Symbolically:
$A \cap B = \emptyset$

Venn Diagram: Circles A and B do not overlap.

(i) How are infinite sets represented in functional programming languages?

(ii) Analyze the use of set theory and binary relations in representing a simple database schema. Give one example and explain the mapping.

(i) Representation of Infinite Sets in Functional Programming Languages

In functional programming languages, **infinite sets** (or more generally, infinite data structures) are typically represented using **lazy evaluation**. Lazy evaluation allows computations to be deferred until their results are actually needed. This enables programmers to define infinite sets without computing all elements at once.

Key Points:

- **Lazy Lists / Streams**: Infinite sets are commonly modeled as infinite lists or streams.
- **Generator Functions**: Functions that recursively define the next element in the set.
- **No memory overflow**: Since only accessed portions are evaluated, infinite structures can be memory-safe.

Example (in Haskell):

haskell

CopyEdit

naturals = [0..] -- Infinite list of natural numbers

evens = [x | x <- naturals, x `mod` 2 == 0] -- Infinite list of even numbers

Here, naturals is an infinite set of numbers starting from 0. Evaluation occurs only as elements are consumed.

(ii) Set Theory and Binary Relations in Database Schemas

Set theory and binary relations are foundational in representing and understanding relational databases.

Key Concepts:

- A **relation** in a database is essentially a **set of tuples**.
- A **table** corresponds to a **relation**, and **rows** correspond to **elements (tuples)** of the set.
- **Binary relations** represent relationships between elements of two sets (e.g., foreign key relationships).

Example Schema:

Let’s model a simple schema:

text

CopyEdit

Students(StudentID, Name)

Courses(CourseID, Title)

Enrollments(StudentID, CourseID)

Mapping to Set Theory:

- Let **S** be the set of all students.
- Let **C** be the set of all courses.
- Let **E** ⊆ S × C be the **binary relation** "is enrolled in".

Thus:

- Students = set of tuples (StudentID, Name)
- Courses = set of tuples (CourseID, Title)
- Enrollments = set of tuples (StudentID, CourseID) = binary relation from Students to Courses

Explanation:

The Enrollments table is a **binary relation** between the set of students and the set of courses. It defines a subset of the Cartesian product S × C. Each tuple in Enrollments means “Student S is enrolled in Course C”.

Benefits:

- Enables use of **relational algebra** (based on set operations) for queries.
- Simplifies understanding of **joins, selections, and projections**.
- Underpins integrity constraints (e.g., referential integrity via subset relations).

Explain the basic concepts of sets with suitable examples. List and define any four set operations. (Note for diagrams u can refer to 1st question!)

A **set** is a **well-defined collection of distinct objects**, considered as an object in its own right. The objects are called **elements or members** of the set.

Key Concepts:

1. **Set Notation**:
    - A set is usually denoted by capital letters (e.g., A, B, C).
    - Elements are listed inside curly brackets {}.
    - Example:
      A = {1, 2, 3, 4}
2. **Element Membership**:
    - If x is an element of set A, we write: $x \in A$
    - If not, we write: $x \notin A$
    - Example:
      $2 \in A$, $5 \notin A$
3. **Types of Sets**:
    - **Finite Set**: Contains a limited number of elements.
      Example: B = {a, e, i, o, u}
    - **Infinite Set**: Unlimited elements.
      Example: N = {1, 2, 3, 4, …}
    - **Empty Set (Null Set)**: Contains no elements, denoted by {} or ∅.
    - **Universal Set (U)**: Contains all possible elements under discussion.
4. **Subset**:
    - A set A is a subset of B ($A \subseteq B$) if every element of A is also in B.
      Example: A = {1, 2}, B = {1, 2, 3} → $A \subseteq B$

Four Fundamental Set Operations

1. Union ($A \cup B$)

- Combines all elements from both sets without duplication.
- **Definition**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$

- **Example**:
  $A = \{1, 2, 3\}$, $B = \{3, 4, 5\} \rightarrow A \cup B = \{1, 2, 3, 4, 5\}$

---

2. Intersection ($A \cap B$)

- Includes only the common elements of both sets.
- **Definition**: $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$
- **Example**:
  $A = \{2, 4, 6\}$, $B = \{4, 6, 8\} \rightarrow A \cap B = \{4, 6\}$

---

3. Difference ($A - B$)

- Elements in A that are not in B.
- **Definition**: $A - B = \{x \mid x \in A \text{ and } x \notin B\}$
- **Example**:
  $A = \{1, 2, 3, 4\}$, $B = \{3, 4, 5\} \rightarrow A - B = \{1, 2\}$

---

4. Complement (A^c)

- Elements not in A, assuming a universal set U.
- **Definition**: $A^c = \{x \in U \mid x \notin A\}$
- **Example**:
  $U = \{1, 2, 3, 4, 5\}$, $A = \{2, 4\} \rightarrow A^c = \{1, 3, 5\}$

(i) What is a model in propositional logic? Give a simple example.

(ii) Apply the concepts of entailment and validity to compare the two. Illustrate with examples.

(i) What is a Model in Propositional Logic?

In **propositional logic**, a **model** is a specific assignment of **truth values** (True or False) to **propositional variables**. A model helps us evaluate whether a **propositional formula** is **true or false** under that assignment.

◇ **Example:**

Let's consider propositional variables:

- **P**: "It is raining"
- **Q**: "The ground is wet"

Let the formula be:
P → Q (If it is raining, then the ground is wet)

Now consider a **model M** where:

- **P = True**
- **Q = True**

Under this model, **P → Q** is **True**, because in logic, an implication is true when both the premise and conclusion are true.

(ii) Entailment vs. Validity

These are key concepts in **logical reasoning** and **truth inference**.

---

1. Entailment (⊨)

- **Definition**: A set of premises **entails** a conclusion if **in every model** where the premises are true, the conclusion is also true.
- **Notation**:
  If **Γ** is a set of premises and **φ** is a conclusion, then:
  Γ ⊨ φ
- **Example**:
  Let Γ = {P, P → Q}

Then Q is entailed: **$\Gamma \models Q$**
Because in **every model** where P is true and $P \rightarrow Q$ is true, Q must also be true.

---

2. Validity

- **Definition**: A propositional formula is **valid** if it is true in **all possible models**.
- **Notation**:
  A formula φ is valid: **$\models \varphi$**
- **Example**:
  The formula: **$(P \vee \neg P)$**
  This is valid because it is **always true**, regardless of whether P is true or false.

🔁 Comparison using Examples

Concept	Description	Example	Result
Entailment	Truth of conclusion follows from premises	$\{P, P \rightarrow Q\} \models Q$	True (inferred conclusion)
Validity	Always true under every model (a tautology)	$(P \vee \neg P)$	Valid formula ($\models \varphi$)

Illustrate the process of converting any formula into Conjunctive Normal Form with an example.

A formula is in **CNF** if it is a **conjunction (AND, ∧)** of **disjunctions (OR, ∨)** of **literals** (atomic propositions or their negations).

Example CNF structure:

$$(A \vee \neg B) \wedge (C \vee D \vee \neg E)$$

🎯 Steps to Convert Any Propositional Formula into CNF

Let's illustrate with an example:

Given Formula:

$$(P \rightarrow Q) \wedge (\neg Q \vee R)$$

☑ Step 1: Eliminate Implications and Biconditionals

Use the equivalence:

- $P \rightarrow Q \equiv \neg P \vee Q$

Apply to the formula:

$(\neg P \vee Q) \wedge (\neg Q \vee R)$

☑ Step 2: Move NOTs Inward (Use De Morgan's Laws if necessary)

Already in proper negation form; nothing to simplify here.

☑ Step 3: Distribute OR over AND if necessary

Our formula is:

$(\neg P \vee Q) \wedge (\neg Q \vee R)$

This is already in CNF! It's a conjunction of two disjunctions (clauses).

🧠 Let's Try a More Complex Example

New Formula:

$((P \wedge Q) \rightarrow R)$

☑ **Step 1: Eliminate Implication**

$(P \land Q) \to R \equiv \neg(P \land Q) \lor R$

So:
$\neg(P \land Q) \lor R$

☑ **Step 2: Apply De Morgan's Law**

$$\neg P \lor \neg Q \lor R$$

☑ **Step 3: Confirm CNF Structure**

$\neg P \lor \neg Q \lor R$

This is already a disjunction of literals — so it's a single clause, and hence in CNF.

□ **Final Summary of CNF Conversion Process:**

1. **Eliminate implications/biconditionals**:
   - $A \to B \Rightarrow \neg A \lor B$
   - $A \leftrightarrow B \Rightarrow (A \land B) \lor (\neg A \land \neg B)$
2. **Move NOTs inward** using De Morgan's laws:
   - ¬(A ∧ B) → ¬A ∨ ¬B
   - ¬(A ∨ B) → ¬A ∧ ¬B
3. **Apply distributive laws**:
   - Distribute OR over AND to get CNF:
     - $A \lor (B \land C) \equiv (A \lor B) \land (A \lor C)$
4. **Result**: A conjunction of disjunctions of literals.

Result: A conjunction of disjunctions of literals.

Explain the concepts of satisfiability, validity, and entailment in propositional logic with suitable examples.

◇ 1. Satisfiability

Definition:

A propositional formula is **satisfiable** if there exists **at least one model (truth assignment)** in which the formula is **true**.

☑ Key Point:

- **At least one** truth assignment makes the formula true.

Example:

Let φ = P∨Q

- If **P = False**, **Q = True** → φ = True ☑
- Hence, φ is **satisfiable** because it is true in at least one model.

❌ Unsatisfiable Formula:

Let φ = P∧¬P

- No truth assignment can make this formula true → φ is **unsatisfiable**

◇ 2. Validity

Definition:

A formula is **valid** if it is **true in all possible models** (i.e., under every truth assignment).

☑ Key Point:

- The formula is **always true** → also called a **tautology**

Example:

Let φ = P∨¬P

- No matter the truth value of P, φ is always true.
- Therefore, φ is **valid**.

Relation to Satisfiability:

- All **valid formulas** are **satisfiable**, but not all satisfiable formulas are valid.

◇ 3. Entailment (⊨)

Definition:

A set of formulas **Γ** *entails* a formula **φ** if **in every model where all formulas in Γ are true**, φ is also true.

☑ **Notation:**

- $\Gamma \vDash \varphi$ means φ logically follows from Γ.

Example:

Let:

- $\Gamma = \{P, P \rightarrow Q\}$
- $\varphi = Q$

Check:

- If **P is true** and **P → Q is true**, then **Q must be true**.

Thus,
$\Gamma \vDash Q$

Important Note:

- Entailment means **truth preservation**: if premises are true, conclusion must be true.
- Entailment does **not** require φ to be valid or always true—just true **whenever the premises are true**.

🧠 **Summary Table**

Concept	Definition	Truth Requirement	Example	Result
Satisfiability	Exists a model where formula is true	At least one true case	PVQP ∨ QPVQ	☑ Satisfiable
Validity	True in all models	Always true	PV¬P	☑ Valid
Entailment	Conclusion follows from premises	Conclusion true if premises are true	From {P, P → Q}, infer Q	☑ Q is entailed

PART – C

Apply logic and function theory to construct a tree structure that supports decision-making algorithms.

Role of Propositional and Predicate Logic in Decision Trees

Propositional Logic:

Propositional logic involves simple statements that can either be true or false. In decision trees, each internal node can be thought of as a **propositional statement or condition**.

Example:

- Let P = "Temperature > 30°C"
- Then:
    - If P is **true**, the decision moves to one subtree.
    - If P is **false**, it moves to another.

Using logic, the path from the root to any leaf represents a **conjunction** of propositions (ANDed statements), which together define the **rule** that leads to that particular decision.

Predicate Logic:

While propositional logic is limited to true/false values, **predicate logic** introduces **quantifiers and variables**, allowing for more expressive decision rules.

For example:

- Predicate: HighRisk(x) might be defined as:
    - $\text{HighRisk}(x) \leftrightarrow (\text{Age}(x) > 60 \land \text{BP}(x) > 140)$

This logic allows decision nodes to work with **attributes of entities** (e.g., x being a person) and to apply **logical inference** to deduce conclusions.

Functional Representation in Tree Structures

Function theory supports tree construction by interpreting decision-making as a series of **function evaluations**. Each node in the tree can be associated with a **decision function** that maps input features to a branch.

Functions in Trees:

Let:

- $f_1(x)$ = function at node 1
- $f_2(x)$ = function at node 2 (depending on f_1's result), and so on

Each function operates on the input space and returns a logical value (typically Boolean), which determines the next step in the decision tree.

Example:
Let x = {Temperature = 35, Humidity = 90}
Define:

- $f_1(x)$ = (x.Temperature > 30)
- $f_2(x)$ = (x.Humidity > 80)

The decision tree would evaluate f_1, and if true, then evaluate f_2, leading to a final decision based on both.

This aligns with the concept of **function composition** and **conditional branching**, foundational in function theory.

Constructing the Decision Tree: A Step-by-Step Logic-Based Approach

The construction of a decision tree using logic and functions can be broken down into the following steps:

Step 1: Define the Universe of Discourse

Identify all input variables or features relevant to the decision. For instance, in a medical diagnosis system:

- Inputs = {Age, Temperature, Blood Pressure, Symptoms, Test Results}

These become the **domain** for logical predicates and decision functions.

Step 2: Establish Logical Conditions (Predicates)

Convert domain knowledge into logical rules.

Examples:

- P_1: Age(x) > 60
- P_2: BP(x) > 140
- P_3: Fever(x) = true

These rules form the **conditions** that internal nodes of the tree will evaluate.

Step 3: Translate Logical Conditions into Functions

Create functions for each logical predicate:

- $f_1(x)$ = (x.Age > 60)
- $f_2(x)$ = (x.BP > 140)
- $f_3(x)$ = (x.Fever = true)

Each function returns a Boolean, enabling branching.

Step 4: Build the Tree Based on Functional Evaluation

Start from the root and recursively apply decision functions:

- At the root: evaluate $f_1(x)$
    - If True: go to left child node
    - If False: go to right child node
- Continue until a terminal condition (leaf node) is reached

Step 5: Label the Leaf Nodes with Actions or Decisions

Each path from root to leaf corresponds to a **decision rule** formed by a conjunction of conditions. Assign outcomes accordingly.

Example rule:

- If Age > 60 ∧ BP > 140 ∧ Fever = true, then **High Risk**

Example: Constructing a Tree for Disease Risk Classification

Let's build a small decision tree for determining if a patient is at **low**, **medium**, or **high** risk of a disease.

Inputs:

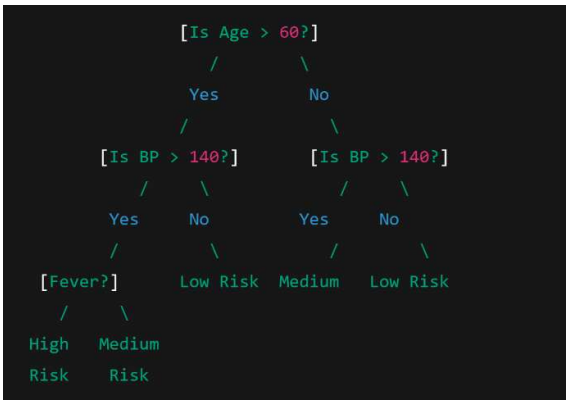
- Age
- Blood Pressure (BP)
- Fever

Logic-Based Rules:

1. If Age > 60 ∧ BP > 140 ∧ Fever = true → High Risk
2. If Age ≤ 60 ∧ BP > 140 → Medium Risk
3. If BP ≤ 140 ∧ Fever = false → Low Risk

Tree Construction:

Each internal node is a **decision function**, and the paths are **logical conjunctions**. The final decisions (leaf nodes) are the result of evaluating those functions in sequence.



Analyze the importance of relations and functions in the formulation of data structures. Explain with suitable diagrams or examples.

1. Definitions: Relations and Functions

- **Relation:**

A **relation** in mathematics is a subset of the **Cartesian product** of two or more sets. It defines how elements of one set are associated with elements of another.

- Mathematically:
  If A and B are sets, a **relation R** from A to B is a subset of A×B.
- Example:
  Let A = {1, 2, 3}, B = {a, b}, then R = {(1, a), (2, b)} is a relation.

- **Function:**

A **function** is a special type of relation where **each element of the domain maps to exactly one element in the codomain**.

- Mathematically:
  f:A→B, such that for every a∈A, there exists exactly one b∈ B with (a,b)∈f.
- Example:
  f(x) = x² maps each integer to its square.

---

2. Role of Relations in Data Structures

Relations are vital in expressing connections between elements. These connections are fundamental in structures such as:

- **Graphs**

  - A **graph** is a data structure built on **binary relations** between nodes (vertices).
  - A graph G=(V,E) where:
    - V is a set of vertices (nodes)
    - E⊆V× V is a **relation** representing edges (directed or undirected)

Use Case: Representing social networks, road maps, dependencies in scheduling.

- **Relational Databases**

  - Tables in databases rely on **relations**:
    - Rows = tuples

- Columns = attributes (sets)
- Foreign keys = define **relational integrity** between tables

Example:
An Employee table related to a Department table via a department_id foreign key defines a **many-to-one relation** from Employee to Department.

◇ **Trees**

- A **tree** is a special case of a graph, with a **parent-child relation**.
- Each node (except root) has one incoming edge (relation) from its parent.
- Binary trees impose functions to left/right child:
  parent→(left_child,right_child)

---

3. Role of Functions in Data Structures

Functions are used to define **access patterns**, **structure behavior**, and **transformation of data**.

◇ **Arrays and Indexing**

- Arrays use a **function from indices to values**:
  - f(i)=array[i], where i is an integer
- This functional mapping provides **constant-time access**, a major strength of arrays.

◇ **Hash Tables**

- A **hash function** maps keys to indices:
  - h:Keys→Z_n (modular space)
  - Collision resolution (e.g., chaining) may use relations or additional functions.

Example:
If h("Alice") = 3, "Alice" is stored at index 3 in the table.

◇ **Object-Oriented Structures**

- In classes/objects, **methods** behave like functions:
  - Each method maps input arguments (domain) to output (codomain).
  - **Object references** and class inheritance define relational structures.

---

4. Combined Example: University Database Schema

Let's model a simple university database with the following entities:

- **Students**
- **Courses**
- **Instructors**

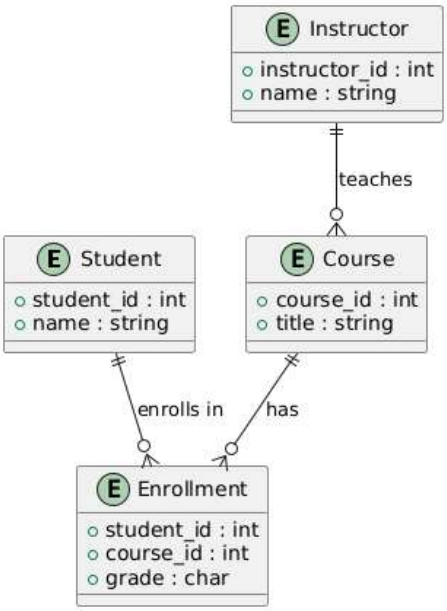
Relations:

- Each **student** can **enroll** in multiple courses → **Many-to-Many**: $R_{enroll} \subseteq Students \times Courses$
- Each **course** is **taught by** one instructor → **One-to-Many**: $R_{teaches} \subseteq Instructors \times Courses$

Functions:

- A function **f** that assigns a **grade**:
  - $f : (Student, Course) \rightarrow Grade$

Diagram Representation



Note this is a just an example diagram read it separately!!!

6. Real-World Significance in Data Structures

- **Linked Lists**: Each node has a function to the next node (next(n) = n+1)
- **Trees**: Parent-to-child relations; left/right children represented by functions
- **Graphs**: Relations modeled as edge sets; important for search and traversal algorithms
- **Heaps**: Functions maintain heap property (e.g., f(parent) > f(child))
- **Stacks/Queues**: Use functions for insert/remove operations with well-defined input/output domains.

Convert the formula ((P ∧ Q) → R) ∨ (¬S) into CNF and DNF. Analyze how each form benefits different logical operations.

I. Original Formula

We are given:

$$((P \land Q) \to R) \lor (\neg S)$$

This formula contains:

- A compound implication $(P \land Q) \to R$
- A disjunction with the negation of a single variable ¬S

II. Step-by-Step Conversion to CNF (Conjunctive Normal Form)

Step 1: Eliminate Implication

Recall:

$$A \to B \equiv \neg A \lor B$$

So:

$$(P \land Q) \to R \equiv \neg(P \land Q) \lor R$$

Hence:

$$((P \land Q) \to R) \lor \neg S \equiv (\neg(P \land Q) \lor R) \lor \neg S$$

Step 2: Apply De Morgan's Laws

$$\neg(P \land Q) \equiv \neg P \lor \neg Q$$

So:

$$(\neg(P \land Q) \lor R) \lor \neg S \equiv ((\neg P \lor \neg Q) \lor R) \lor \neg S$$

This is a long chain of disjunctions, and disjunction is associative:

$$\neg P \lor \neg Q \lor R \lor \neg S$$

✅ CNF Form

The CNF is a **conjunction of disjunctions**. Since the entire formula is already a disjunction, and there is only **one clause**, it is technically in CNF already:

$$\text{CNF: } (\neg P \lor \neg Q \lor R \lor \neg S)$$

This is a single disjunctive clause, hence a valid CNF.

III. Step-by-Step Conversion to DNF (Disjunctive Normal Form)

Step 1: Start from the implication again

$$((P \land Q) \rightarrow R) \lor \neg S \equiv (\neg(P \land Q) \lor R) \lor \neg S$$

We now need to **distribute** disjunctions and create a **disjunction of conjunctions**.

$$\neg(P \land Q) \lor R \lor \neg S \equiv (\neg P \lor \neg Q) \lor R \lor \neg S$$

We want to express the formula as a **disjunction of conjunctions**, so we explore all **truth cases**.

But first, we can **reassociate**:

$$(\neg P \lor \neg Q \lor R \lor \neg S)$$

This is already a disjunction of literals. To express it in true **DNF**, we will consider truth table–based expansion.

Step 2: Truth Table Analysis (Exhaustive)

We consider all possible combinations of truth values for P, Q, R, and S and determine for which combinations the formula evaluates to **true**. From there, we derive the corresponding **minterms** (conjunctions) that produce true outputs.

The table will have $2^4 = 16$ rows. Below are the truth values that make the formula true, followed by their respective minterms.

P	Q	R	S	Formula	Minterm
F	F	F	F	T	¬P ∧ ¬Q ∧ ¬R ∧ ¬S
F	F	F	T	T	¬P ∧ ¬Q ∧ ¬R ∧ S
F	F	T	F	T	¬P ∧ ¬Q ∧ R ∧ ¬S
F	F	T	T	T	¬P ∧ ¬Q ∧ R ∧ S
F	T	F	F	T	¬P ∧ Q ∧ ¬R ∧ ¬S
F	T	F	T	T	¬P ∧ Q ∧ ¬R ∧ S
F	T	T	F	T	¬P ∧ Q ∧ R ∧ ¬S
F	T	T	T	T	¬P ∧ Q ∧ R ∧ S
T	F	F	F	T	P ∧ ¬Q ∧ ¬R ∧ ¬S
T	F	F	T	T	P ∧ ¬Q ∧ ¬R ∧ S
T	F	T	F	T	P ∧ ¬Q ∧ R ∧ ¬S

P	Q	R	S	Formula	Minterm
T	F	T	T	T	P ∧ ¬Q ∧ R ∧ S
T	T	T	F	T	P ∧ Q ∧ R ∧ ¬S
T	T	T	T	T	P ∧ Q ∧ R ∧ S
T	T	F	F	T	P ∧ Q ∧ ¬R ∧ ¬S
T	T	F	T	F	*Excluded*

Only one combination results in false: when P = T, Q = T, R = F, S = T.

✅ DNF Form

Disjoin all the 15 minterms where the formula evaluates to true:

$$(\neg P \land \neg Q \land \neg R \land \neg S) \lor (\neg P \land \neg Q \land \neg R \land S) \lor \cdots \lor (P \land Q \land \neg R \land \neg S) \lor (P \land Q \land R \land \neg S) \lor (P \land Q \land R \land S)$$

This is a long DNF expression but it is **logically complete** and reflects the definition of DNF: **a disjunction of conjunctions of literals.**

IV. Comparative Analysis: CNF vs. DNF

🔷 CNF (Conjunctive Normal Form)

- **Form**: Conjunction of disjunctions.
- **Example**:

$$(\neg P \lor \neg Q \lor R \lor \neg S)$$

- **Use Cases**:
  - Used in **SAT solvers** (e.g., DPLL, CDCL).
  - Efficient in proving **unsatisfiability**.
  - Practical for **constraint satisfaction problems**.
- **Benefits**:
  - Short, compact.
  - Suitable for **automated theorem proving**.
- **Drawbacks**:
  - May lose readability for representing **explicit solutions**.

🔶 DNF (Disjunctive Normal Form)

- **Form**: Disjunction of conjunctions.
- **Example**:

$$(\neg P \land \neg Q \land R \land \neg S) \lor \ldots \lor (P \land Q \land R \land S)$$

- **Use Cases**:
  - Excellent for expressing **explicit satisfying models**.
  - Used in **pattern recognition**, **digital circuits**, and **logic programming**.
- **Benefits**:
  - Highlights all **models that satisfy** the formula.
  - Useful in **enumerating solutions** or for exhaustive search.
- **Drawbacks**:
  - Can be **exponentially large**.
  - Inefficient for **large or complex formulas**.

Describe the full process of converting a propositional formula into CNF and applying the DPLL algorithm or SAT solver techniques to check satisfiability. Provide detailed steps with examples.

I. Step-by-Step Conversion to CNF (Conjunctive Normal Form)

✅ What is CNF?

A formula is in **Conjunctive Normal Form (CNF)** if it is a **conjunction** (AND, ∧) of one or more **clauses**, where each clause is a **disjunction** (OR, ∨) of literals (a literal is a variable or its negation).

Example of CNF:

$$(P \vee \neg Q) \wedge (\neg R \vee S \vee T)$$

🔷 General Steps to Convert Any Formula to CNF

1. **Eliminate bi-implications (↔)**
   Replace $A \leftrightarrow B$ with $(A \rightarrow B) \wedge (B \rightarrow A)$
2. **Eliminate implications (→)**
   Replace $A \rightarrow B$ with $\neg A \vee B$
3. **Apply De Morgan's Laws**
   Push negations inward:
   - $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$
   - $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$
4. **Eliminate double negations**
   Replace $\neg(\neg A)$ with A
5. **Distribute disjunction over conjunction**
   Apply distributive law:
   - $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$
6. **Flatten nested conjunctions/disjunctions**
   Normalize structure: multiple conjunctions become a flat AND of clauses.

✅ **Example Conversion**

Let's convert the formula:

$$(P \to (Q \land R)) \lor S$$

Step 1: Eliminate implication

$$P \to (Q \land R) \equiv \neg P \lor (Q \land R)$$

So, the formula becomes:

$$(\neg P \lor (Q \land R)) \lor S$$

Step 2: Flatten with associativity

Since ∨ is associative, group accordingly:

$$\neg P \lor (Q \land R) \lor S$$

Step 3: Distribute ∨ over ∧

Apply:

$$A \lor (B \land C) \equiv (A \lor B) \land (A \lor C)$$

Here, A = ¬P ∨ S, and B = Q, C = R

So:

$$(\neg P \lor Q \lor S) \land (\neg P \lor R \lor S)$$

This is now in **CNF**.

II. Applying the DPLL Algorithm

☑ What is the DPLL Algorithm?

DPLL (Davis–Putnam–Logemann–Loveland) is a **recursive backtracking algorithm** for deciding satisfiability of formulas in CNF. It forms the core of most modern **SAT solvers**.

◇ Key Concepts in DPLL

- **Unit Clause Rule**: If a clause has only one literal, assign a truth value to satisfy it.
- **Pure Literal Rule**: If a variable appears only in positive (or only in negative) form, assign a value that satisfies all such occurrences.
- **Backtracking Search**: Choose a variable, assign a truth value, simplify the formula, and recursively check satisfiability.

✅ **Example Using DPLL**

Consider the CNF formula:

$$(\neg P \lor Q) \land (\neg Q \lor R) \land (\neg R) \land (P)$$

This can be represented as a set of clauses:

- Clause 1: (¬P ∨ Q)
- Clause 2: (¬Q ∨ R)
- Clause 3: (¬R)
- Clause 4: (P)

🔷 Step-by-Step DPLL Process

Step 1: Simplify with unit clause

- Clause 3: (¬R) ⇒ R = False
  Assign: **R = False**

Now update clauses:

- Clause 2: (¬Q ∨ R) → (¬Q ∨ False) = (¬Q)
- New set of clauses:
  (¬P ∨ Q), (¬Q), (P)

Step 2: New unit clause → (¬Q)

- ⇒ Q = False

Update clause 1: (¬P ∨ Q) → (¬P ∨ False) = (¬P)
Remaining clauses: (¬P), (P)

Step 3: Conflict detected

- (¬P), (P) ⇒ contradiction
  Backtrack

Step 4: Backtracking decision

We assumed R = False. Now try **R = True**

Clause 3: (¬R) → False
This clause cannot be satisfied ⇒ contradiction again.

Therefore, the **entire formula is unsatisfiable**.

III. Integration with Modern SAT Solvers

Modern SAT solvers (e.g., MiniSAT, Z3) expand DPLL with:

- **Conflict-Driven Clause Learning (CDCL)**: Learn from failures to prune the search.
- **Non-chronological Backtracking**: Return to earlier decisions rather than step-by-step.
- **Watched Literals**: Improve clause scanning efficiency.

These solvers use **CNF as input** and apply variations of DPLL under the hood to efficiently solve real-world problems in:

- Hardware verification
- AI planning
- Cryptographic protocol checking
- Scheduling and optimization

IV. Summary and Benefits

Aspect	CNF Conversion	DPLL Algorithm
Purpose	Normalizes formula for computation	Solves satisfiability problem on CNF
Output	Set of conjunctive clauses	SAT / UNSAT and model (if SAT)
Benefits	Enables automation, symbolic computation	Efficient decision-making via backtracking
Limitations	May increase formula size	May need heuristics to avoid exponential search