Part – A

Question
An interpretation in predicate logic must specify_____ A. Variable types only B. Truth values only **C. Domain and predicate meanings** D. Inference rules
The formula $\exists x \neg P(x)$ is logically equivalent to _____ **A. $\neg\forall x\ P(x)$** B. $\forall x \neg P(x)$ C. $\neg\exists x\ P(x)$ D. $P(x) \wedge Q(x)$
A universally valid predicate logic sentence is one that_____ A. Is true in some model B. Is not satisfiable **C. Holds in every interpretation** D. Is contradictory
In a pattern sequence, shape color alternates Red → Blue → Red → Blue → _____ A. Blue **B. Red** C. Green D. Yellow
A logic rule says: "If shape is square and blue, mark as target." Identify rule type. **A. Rule-based** B. Genetic C. Heuristic D. Randomized
A missing figure in a sequence most likely involves_____ A. Irregular change **B. Consistent logic** C. Random choice D. Artistic design
A clustering algorithm groups figures based on_____ A. Colors B. Size **C. Feature similarity** D. Random selection
A cube is painted on all sides and cut into smaller cubes. How many cubes have 3 painted faces? A. 6 **B. 8** C. 4 D. 2
Two opposite faces of a cube are red and blue. If red is on top, where is blue? A. Left B. Right **C. Bottom** D. Front
A cube shows 1, 2, 3 on three adjacent faces. The sum of opposite faces is_____ A. 5 **B. 7** C. 9 D. Cannot determine
Identify the symbol used for the universal quantifier in predicate logic? A. $\exists$ B. $\wedge$ **C. $\forall$** D. $\rightarrow$
Which of the following is logically equivalent to: $\neg\exists x\ P(x)$ A. $\exists x \neg P(x)$ B. $\neg\forall x\ P(x)$ **C. $\forall x \neg P(x)$** D. $\forall x\ P(x)$
What is the natural language translation of: $\forall x$ (Student(x) → Studies(x)) A. All who study are students **B. All students study** C. Some students study D. No student studies
Find the transformation method that changes the size of an object without changing its shape? A. Flipping B. Rotation **C. Scaling** D. Translation
In an image sequence, a square shifts from the left edge to the right edge of each frame. Which transformation is illustrated? A. Scaling **B. Translation** C. Rotation D. Morphing
A shape changes from Triangle → Square → Pentagon → Hexagon. What is the rule applied here? A. Reducing angles B. Color change C. Rotation by 90° **D. Adding one side to the shape in each step**

What is the sum of numbers on opposite faces of a standard die? A. 5 B. 6 **C. 7** D. 8
Why is edge detection used in image preprocessing for figure analysis? A. To smooth image color **B. To highlight figure boundaries** C. To increase image resolution D. To rotate the figure
Identify the feature used in pattern recognition? A. Color temperature B. Texture granularity **C. Feature vector** D. Optical lens
Which approach provides the best reasoning strategy for visual figure analysis involving transformations and rule inference? A. Human-centric guessing B. Basic template matching **C. Symbolic and statistical hybrid models** D. Color detection

Part – B

Translate the given database query into predicate logic: “Find all students who have passed."

1. Define Predicates

Let’s assume the following predicates:

- Student(x): x is a student
- Passed(x): x has passed

---

2. Required Logic Translation

We want all entities **x** such that x is a student **and** x has passed.

So, the translation in **predicate logic** is:

∀x [(Student(x) ∧ Passed(x))]
(*This states that every x in the domain is both a student and has passed, but this is too strong unless the query demands it.*)

However, since the query says **“find all students who have passed”**, it is better to express it as a **set of x** that satisfy the condition.

Hence, the correct **predicate logic representation** is:

{ x | Student(x) ∧ Passed(x) }

Or using **existential quantification** for query logic (if we are querying whether such students exist):

∃x [Student(x) ∧ Passed(x)]

Final Translation for Selection Query:

If you're **selecting** such students from a database:

Select x such that Student(x) ∧ Passed(x)
→ { x | Student(x) ∧ Passed(x) }

This logical form represents all individuals **x** in the domain who satisfy both conditions.

(i) State the difference between Universal and Existential quantifiers with an example each.

(ii) Illustrate the decision procedure to check satisfiability of predicate formulas.

(i) Difference Between Universal and Existential Quantifiers

Aspect	Universal Quantifier (∀)	Existential Quantifier (∃)
Symbol	∀	∃
Meaning	Asserts that a property holds for **all elements**	Asserts that a property holds for **at least one element**
Scope	Applies to the **entire domain** of discourse	Applies to **some elements** in the domain
Truth Requirement	Formula must be **true for every instance**	Formula is **true if one instance satisfies** it
Example (English)	"All dogs bark."	"Some dogs bark."
Example (Logic)	$\forall x\ (Dog(x) \rightarrow Bark(x))$	$\exists x\ (Dog(x) \wedge Bark(x))$
Negation Relation	$\neg\forall x\ P(x) \equiv \exists x\ \neg P(x)$	$\neg\exists x\ P(x) \equiv \forall x\ \neg P(x)$

(ii) Decision Procedure to Check Satisfiability of Predicate Formulas

1. **Convert to Prenex Normal Form (PNF)**
   Move all quantifiers to the front of the formula so the logical structure is easier to analyze.
2. **Skolemization**
   Eliminate existential quantifiers by replacing them with Skolem functions, preserving logical equivalence in satisfiability.
3. **Convert to Clause Form (Conjunctive Normal Form)**
   Transform the quantifier-free part of the formula into CNF for resolution-based methods.
4. **Standardize Variables**
   Rename variables to ensure that no two quantifiers use the same variable name.

5. **Apply Herbrand's Theorem**
   Generate Herbrand Universe and Herbrand Base, which are sets of ground terms and ground atoms derived from the formula.

6. **Instantiate and Test**
   Instantiate variables using terms from the Herbrand Universe to create ground instances of the formula.

7. **Apply Resolution or Model Checking**
   Use resolution refutation or truth assignment in the ground domain to check if the formula is satisfiable (i.e., has at least one model).

i) Define non-verbal reasoning in the context of problem-solving.

(ii) Apply a pattern recognition algorithm to detect anomalies in a shape animation sequence.

(i) Definition of Non-Verbal Reasoning in the Context of Problem-Solving

Non-verbal reasoning is the ability to understand and solve problems using visual information rather than words or numbers. It involves interpreting patterns, shapes, sequences, spatial relationships, and diagrams to make logical inferences. Unlike verbal reasoning, it does not rely on language skills, making it a universal measure of cognitive ability. This skill is crucial in assessing general intelligence, designing AI systems for visual recognition, and solving abstract reasoning tests.

In problem-solving, non-verbal reasoning helps individuals or systems detect regularities, transformations, or inconsistencies in visual data. This includes tasks such as identifying the next shape in a series, completing a matrix with missing elements, or recognizing mirrored or rotated images. It is frequently used in IQ tests, competitive exams, machine vision, robotics, and image-based decision systems.

Example:
Consider a pattern sequence where a square rotates 90° clockwise in each successive image. The problem may ask, "Which figure comes next?" A person (or algorithm) must analyze the rotation rule, predict the next orientation, and select the correct figure from given options. No text or numerical data is used—only spatial and visual logic is applied.

(ii) Application of a Pattern Recognition Algorithm to Detect Anomalies in a Shape Animation Sequence

To detect anomalies in a shape animation sequence, a pattern recognition pipeline can be implemented using the following steps:

1. **Frame Extraction**
   Break the animation into individual image frames that represent each step in the sequence.
2. **Preprocessing**
   Apply grayscale conversion, noise reduction, and edge detection (e.g., using the Canny algorithm) to highlight shape boundaries.
3. **Feature Extraction**
   Use **Hu Moments** or **Histogram of Oriented Gradients (HOG)** to capture rotation- and scale-invariant features of each shape in the sequence.
4. **Sequence Pattern Modeling**
   Model the shape progression using a known transformation pattern (e.g., consistent rotation, translation, or addition of sides). Represent each frame as a vector of extracted features.
5. **Anomaly Detection Algorithm**
   Use unsupervised learning techniques like **Autoencoders** or **One-Class SVM**, or

compare using **Euclidean distance** between feature vectors of successive frames to identify irregular jumps or inconsistencies.

6. **Comparison with Expected Pattern**
   Compare actual transitions between frames with expected transformations. If a frame violates the learned transformation rule (e.g., incorrect rotation angle or unexpected shape), it is flagged as an anomaly.

7. **Visualization and Interpretation**
   Highlight or extract the anomalous frame(s) for further analysis. This helps in validating test quality or training AI systems to mimic human visual reasoning.

Compare the Naive and Computational approaches to solving Cube and Dice problems.

Aspect	Naive Approach	Computational Approach
Definition	Solves problems using **intuition, observation, and basic rules**	Uses **systematic rules, symbols, and formal logic** to represent and solve configurations
Used By	Beginners or during initial learning phases	Advanced learners, test-takers under time pressure, or automated systems
Methodology	Visualizes cube rotations, face relationships mentally	Encodes faces, positions, and relationships numerically or symbolically
Accuracy	Prone to **errors** in complex configurations	High accuracy due to step-by-step deduction
Speed	Fast for **simple** problems	Slightly slower initially but **more consistent** as complexity increases
Handling Rotation	Relies on mentally imagining rotations, may confuse left/right orientation	Uses structured matrices or mapping rules to simulate rotation
Memory Dependency	High – must remember prior positions or labels	Lower – uses notes, tables, or systematic face-pair tracking
Face Relationships	Observed visually (e.g., opposite faces can't be adjacent)	Tabulated or coded (e.g., opposite face pairs: 1–6, 2–5, etc.)
Common Tools Used	Human memory, visualization, paper folding mental model	Tables, diagrams, matrices, logical pair mapping
Example Problem	"If 1 is opposite 6 and 2 is adjacent to 3, what is opposite to 2?" – guessed visually	Same question is broken into steps: list all visible faces, apply known opposites, deduce missing ones
Handling Complex Scenarios	Difficult to manage 3D mental rotations	Handles complex dice and cube unfolding with greater clarity

Aspect	Naive Approach	Computational Approach
Suitability for Automation	Not suitable for automation	Ideal for automation (can be coded using rules and condition checks)

Apply the concept of predicate logic to translate the following English sentences into well-formed formulas. Also state whether they are satisfiable.

Sentences:

a) “All humans are mortal.”

b) “Some cats do not like water.”

(a) "All humans are mortal."

Translation into Predicate Logic:
Let:

- Human(x) = "x is a human"
- Mortal(x) = "x is mortal"

Then the logical form is:
∀x (Human(x) → Mortal(x))
("For all x, if x is a human, then x is mortal.")

Satisfiability:
Yes, this formula is **satisfiable**. It is satisfied in any interpretation where every object that satisfies Human(x) also satisfies Mortal(x). It is **not logically false**—it becomes false only if there exists a human who is not mortal. So it's satisfiable and also **valid** if interpreted universally.

(b) "Some cats do not like water."

Translation into Predicate Logic:
Let:

- Cat(x) = "x is a cat"
- Likes(x, Water) = "x likes water"
- Water = a constant representing water

Then the logical form is:
∃x (Cat(x) ∧ ¬Likes(x, Water))
("There exists at least one x such that x is a cat and x does not like water.")

Satisfiability:
Yes, this formula is **satisfiable**. It is satisfied in any interpretation where **at least one cat exists that does not like water**. It is **not valid** (i.e., not true in all interpretations), but definitely satisfiable in realistic models—like in real life, where many cats dislike water.

Summary:

- (a) ∀x (Human(x) → Mortal(x)) → **Satisfiable** (and also valid)

- (b) $\exists x\ (Cat(x) \land \neg Likes(x, Water)) \rightarrow$ **Satisfiable** (but not valid)

Compare and evaluate different approaches to pattern recognition

1. Template Matching

Template matching is a straightforward method where the input pattern is compared against stored templates (reference patterns) to find the best match.

Working Principle:

It uses similarity measures such as correlation or pixel-wise comparison. The input image is matched directly with one or more templates under different transformations (like scale or rotation).

Example: Matching handwritten digits to pre-defined images of each digit in postal code recognition.

Advantages:

- Simple to implement.
- Effective when there's little variation in size, orientation, or noise.

Limitations:

- Highly sensitive to scale, rotation, and noise.
- Requires large storage and computation for template databases.

2. Statistical Classification

This method models patterns as points in a feature space and uses statistical techniques to assign them to predefined classes.

Working Principle:

Features are extracted from input data (e.g., size, shape, intensity), and classifiers like Bayesian, k-NN, or decision trees use probability distributions to classify the input.

Example: Classifying plant species based on measured attributes like petal length, sepal width, etc.

Advantages:

- Handles variation and noise well.
- Can model uncertainty and use probabilistic reasoning.

Limitations:

- Requires large labeled datasets for accurate modeling.
- Performance depends heavily on feature selection and statistical assumptions.

3. Syntactic (Structural) Matching

This approach treats patterns as structures made of sub-patterns (primitives), similar to grammar rules in language processing.

Working Principle:
Patterns are described using formal grammars, and recognition involves parsing input using syntactic rules. It emphasizes the relationship between parts of the pattern.

Example: Recognizing chemical structures or circuit diagrams based on how their components are arranged.

Advantages:

- Excellent for complex patterns with well-defined structures.
- Allows hierarchical and recursive pattern recognition.

Limitations:

- Computationally intensive parsing.
- Requires accurate structural decomposition and grammar definition.

4. Neural Networks
Neural networks are data-driven models that learn patterns through training by adjusting internal parameters (weights) based on error feedback.

Working Principle:
Inputs are passed through layers of interconnected artificial neurons. The network learns to classify patterns by minimizing prediction error via backpropagation.

Example: Facial recognition using convolutional neural networks (CNNs) that learn from thousands of face images.

Advantages:

- High accuracy and adaptability.
- Effective for complex, non-linear, and high-dimensional data.

Limitations:

- Requires large datasets and significant computational resources.
- Acts as a "black box," often lacking interpretability.

Explain about various pattern recognition algorithms

THIS ONE IS THE SAME AS THE PREVIOUS QUESTION LITERALLY!

Explain the different types of clustering algorithms used in pattern recognition

Clustering Type	Working Principle	Example Application	Advantages	Disadvantages
Centroid-Based (K-Means)	Partitions data into *k* clusters by minimizing intra-cluster distance to centroids.	Customer segmentation based on purchase behavior.	Simple, fast for large datasets, easy to implement.	Requires *k*, sensitive to initial centroids, fails with non-spherical clusters.
Density-Based (DBSCAN)	Forms clusters based on areas of high point density, separates low-density noise.	Geolocation hotspot detection or object grouping.	Finds arbitrarily shaped clusters, handles noise, no need for *k*.	Fails with varying densities, sensitive to eps and minPts parameters.
Distribution-Based (GMM)	Assumes data is from a mixture of distributions; uses EM to fit and assign probabilities.	Speaker identification in audio signal processing.	Models elliptical/overlapping clusters, soft assignment.	Assumes distribution shape, computationally expensive, can overfit.
Hierarchical	Builds nested clusters via agglomeration (bottom-up) or division (top-down).	Gene clustering or taxonomy creation in bioinformatics.	No need for *k*, reveals cluster hierarchy via dendrogram.	Slow for large data, sensitive to linkage criteria and distance measures.

Part – C

i) Show the equivalence of ∃x ¬P(x) and ¬∀x P(x) using truth and semantic analysis.

(ii) Demonstrate how pattern recognition algorithms work for missing frame prediction

(iii) Solve a figure formation puzzle using rule deduction. Explain each inference step.

(i) Equivalence of ∃x ¬P(x) and ¬∀x P(x)

To prove that the formulas **∃x ¬P(x)** and **¬∀x P(x)** are logically equivalent, we can use **semantic analysis** (truth-based reasoning):

Statement 1: ∃x ¬P(x)

There exists at least one element **x** in the domain such that **P(x)** is **not true**.

Statement 2: ¬∀x P(x)

It is **not true** that **P(x)** holds for **all elements** in the domain.

Truth Table-Based Semantic Analysis:

Let's assume a domain D = {a, b, c} and define P(x) on this domain:

x	P(x)
a	True
b	True
c	False

Now:

- **∀x P(x)** = True only if P(a), P(b), and P(c) are all true ⇒ Here, it's **False**
- So, **¬∀x P(x)** = True
- And since P(c) is False, then **∃x ¬P(x)** is also True

Thus, in any model:

- If **∃x ¬P(x)** is true, then **¬∀x P(x)** is also true
- If **¬∀x P(x)** is true, that means not all x satisfy P(x), so at least one x must fail ⇒ ∃x ¬P(x) is true

Hence, **∃x ¬P(x) ≡ ¬∀x P(x)** is semantically and logically equivalent.

(ii) Pattern Recognition Algorithms for Missing Frame Prediction

To predict missing frames in a visual pattern or animation sequence, pattern recognition algorithms analyze transformations across existing frames and infer the likely next or missing element.

Steps:

1. **Frame Input Collection**
   Extract the existing frames (e.g., images of geometric shapes in sequence).
2. **Preprocessing**
   Apply grayscale conversion, resizing, and noise reduction to standardize input.
3. **Feature Extraction**
   Use Hu Moments (for shape invariance) or HOG (for edge orientation) to convert shapes into numerical feature vectors.
4. **Pattern Modeling**
   Model the transformation logic using sequence rules—e.g., geometric progression, color alternation, or rotation angles.
5. **Prediction Model**
   Use machine learning models (like RNNs or LSTMs for sequential learning) or rule-based logic (e.g., rotate shape by +90° each frame) to estimate the missing frame's features.
6. **Reconstruction**
   Convert the predicted feature vector back into an image representation using shape synthesis or retrieval from a shape dataset.

Example:
If the sequence shows a triangle rotating 90° per frame, and the third frame is missing, the algorithm uses frames 1 and 2 to deduce the pattern (e.g., clockwise rotation) and reconstruct the missing third frame based on expected transformation.

(iii) Solving a Figure Formation Puzzle Using Rule Deduction

Problem Example:

You are given a sequence:

- Figure 1: Circle
- Figure 2: Circle with a dot
- Figure 3: Circle with two dots
- Figure 4: ?

Step-by-Step Inference:

1. **Observe Structural Changes**
   Each figure retains a circular shape but **adds one dot** in each step.
2. **Deduce the Rule**
   Rule: Each new figure = Previous figure + 1 additional dot

3. **Apply Rule to Find Next Figure**
   Since Figure 3 has two dots, Figure 4 must have **three dots inside the circle**
4. **Conclusion**
   The next figure is a **circle with three internal dots**.

Reasoning Type:

This is a **rule-based sequential pattern**, where the visual change is additive and consistent.

(i) Given ∀x(Student(x) → Smart(x)), prove that if John is a student, then he is smart.

(ii) Construct a logic system to identify animation frame errors based on symmetry rules.

(iii) Apply clustering and logical reasoning together to solve a mixed-pattern figure case study.

(i) Prove: If ∀x(Student(x) → Smart(x)), and John is a student, then John is smart

We are given:

1. **∀x(Student(x) → Smart(x))** – For all x, if x is a student, then x is smart
2. **Student(John)** – John is a student
   We are to prove: **Smart(John)**

Proof (Using Universal Instantiation and Modus Ponens):

- From the universal statement, we instantiate it for a specific individual:

From ∀x(Student(x) → Smart(x)), we get Student(John) → Smart(John)

- Given: Student(John)
- Apply **Modus Ponens**:

If Student(John) → Smart(John) and Student(John) is true, then Smart(John) must be true.

Therefore, Smart(John) is logically proven based on predicate logic inference.

(ii) Constructing a Logic System to Identify Animation Frame Errors Based on Symmetry Rules

To detect errors in animation sequences where symmetry is expected, the logic system involves formalizing shape properties and applying rule-checking:

Components of the Logic System:

1. **Frame Representation:**
   Each animation frame is treated as a set of geometric elements with measurable properties (e.g., orientation, symmetry axis, centroid alignment).
2. **Predicate Definitions:**
   - Symmetric(x): True if object x is symmetric across a defined axis.
   - Aligned(x, y): True if shapes x and y are centrally or rotationally aligned.
   - Rotation(x, θ): True if object x is rotated by angle θ from the reference shape.
3. **Symmetry Rule Encoding:**
   - ∀x ∈ Frames: Symmetric(x) ∧ Aligned(x, x_prev)

- All shapes in frames must follow the same symmetry transformation (e.g., mirror symmetry or 90° rotation).

4. **Anomaly Detection:**
    - A frame $\mathbf{F_i}$ violates the rule if $\neg\text{Symmetric}(F_i) \vee \neg\text{Aligned}(F_i, F_{i-1})$
    - Such frames are flagged as errors or outliers.
5. **Automation Tools:**
    - Use **HOG features** or **Zernike Moments** to check symmetry.
    - Apply rules via logic scripts or symbolic reasoning engines (e.g., Prolog or rule-based engines in Python).

Example:
If frame 1 and frame 2 both show a square reflected across the Y-axis, and frame 3 shows an asymmetric rotated triangle, then frame 3 violates the symmetry rule.

(iii) Applying Clustering and Logical Reasoning Together to Solve a Mixed-Pattern Figure Case Study

In a mixed-pattern visual reasoning problem (e.g., a 3x3 matrix of figures), different rows or columns follow different rules. Combining clustering and logic helps isolate patterns and deduce missing or incorrect figures.

Case Study Setup:

You are given a 3×3 grid of images. Each row follows a different transformation rule (e.g., rotation, size change, shape addition), but one figure is missing.

Step-by-Step Approach:

1. **Feature Extraction:**
    - Extract features using **Hu Moments** or **Zernike Moments** for rotation/scaling invariance.
    - Represent each figure as a vector.
2. **Clustering (Unsupervised):**
    - Apply **DBSCAN** to group figures that follow similar transformations (e.g., rotation cluster, scaling cluster).
    - Each cluster corresponds to a distinct rule type.
3. **Logical Rule Deduction:**
    - For each cluster, analyze the common transformation logic:
        - Cluster A → shape rotates 90°
        - Cluster B → shape size doubles
        - Cluster C → shape count increases by 1

4. **Matrix Rule Assignment:**
    - Assign identified clusters to rows/columns in the matrix.
    - Use logical inference to predict the rule continuation for the missing figure.
5. **Prediction:**
    - If row 2 is assigned to Cluster A (rotation), and the first two figures are rotated by 90° and 180°, then the third must be 270°.
    - The model synthesizes or selects this figure from the shape dataset.

Benefits of Combined Approach:

- **Clustering** filters patterns from noisy or mixed data.
- **Logical reasoning** enforces rule-based constraints.
- This hybrid method ensures accurate detection even in complex visual puzzle grids.

Analyze how predicate logic is applied in Natural Language Processing (NLP) and Databases. Use examples to support your analysis.

Predicate logic (also known as First-Order Logic or FOL) is a formal system in logic that uses quantified variables over non-logical objects and allows the expression of statements involving predicates and relations. It is more expressive than propositional logic, enabling the representation of objects, their properties, and relationships between them using functions, constants, predicates, variables, and quantifiers like ∀ (for all) and ∃ (there exists).

Working of Predicate Logic in NLP

Semantic Representation of Sentences
In NLP, predicate logic allows the conversion of natural language into formal structures. For example,

- Sentence: *"John likes Mary"*
- Logical form: Likes(John, Mary)
  This represents the action (Likes) and the entities involved, enabling semantic parsing and reasoning.

Disambiguation of Sentences
Natural language is often ambiguous. Predicate logic helps resolve multiple meanings.
Example: *"Every student read a book."*

- $\forall x(\text{Student}(x) \rightarrow \exists y(\text{Book}(y) \land \text{Read}(x, y)))$ → different book per student
- $\exists y(\text{Book}(y) \land \forall x(\text{Student}(x) \rightarrow \text{Read}(x, y)))$ → same book for all students

Inference and Knowledge Derivation
Predicate logic supports deductive reasoning.
Example:

- $\forall x(\text{Human}(x) \rightarrow \text{Mortal}(x))$
- Human(Socrates)
- Therefore, Mortal(Socrates)

Question Answering in NLP Systems
Questions can be converted into logical queries.

- Question: *"Who founded Google?"*
- Logic: $\exists x(\text{Founded}(x, \text{Google}))$
  The system searches a knowledge base to return the correct value (e.g., Larry Page or Sergey Brin).

Handling Negation and Quantifiers
NLP systems must understand negation and scope.

- Sentence: *"No cat can bark."*

- Logical Form: ∀x(Cat(x) → ¬CanBark(x))
  This avoids misinterpretation of negative statements.

Semantic Role Labeling and Intent Modeling
Predicate logic helps determine *who* did *what* to *whom*.

- Sentence: *"Alice sent a message to Bob."*
- Logic: Sent(Alice, Message, Bob)
  Useful in chatbots and voice assistants to identify actions and targets.

Coreference Resolution
Predicate logic uses variables to maintain consistency in references.

- Sentences: *"Emma entered the room. She sat down."*
- Logic: Entered(x, Room) ∧ SatDown(x) where x = Emma

Working of Predicate Logic in Databases

Relational Model Foundations
Predicate logic forms the mathematical basis for relational databases. Tables can be seen as predicates, and each row as a fact.

- Table: Students(name, course)
- Predicate: Student(x, y) → means "x is enrolled in y"

Formulating Queries
SQL queries are logical expressions.

- Query: *Find students who passed*
- Logic: ∃x(Student(x) ∧ Passed(x))
  SQL Equivalent: SELECT name FROM Students WHERE status = 'Passed'

Integrity Constraints
Predicate logic can express rules that must hold true.

- Rule: *Every student must be enrolled in at least one course*
- Logic: ∀x(Student(x) → ∃y(Enrolled(x, y)))

View Definition and Optimization
Views can be represented using logical expressions for better query planning and optimization.

- Logic: View(x) ↔ Student(x) ∧ Score(x, y) ∧ y > 80

Trigger and Rule-Based Actions
Triggers in databases can be modeled using logical conditions.

- If Score(x) < 40, then Alert(x) is activated.

Data Validation and Consistency Checking
Predicate logic is used to check whether data adheres to specified logical conditions, preventing anomalies.

Data Mining and Pattern Matching
Logical patterns (e.g., ∃x, y such that x bought item y more than 3 times) can drive recommendation engines or analytics modules in large databases.

Advantages of Predicate Logic in NLP and Databases

- **Expressiveness**
  Predicate logic can express complex sentences involving relationships, rules, negation, and quantification, enabling richer representation than propositional logic.
- **Precision and Formality**
  Predicate logic provides a clear, formal syntax and semantics, reducing ambiguity in meaning – crucial for machines to interpret language or data accurately.
- **Support for Reasoning and Inference**
  Systems based on predicate logic can deduce new information, verify consistency, and explain results – enhancing explainability in NLP and database decisions.
- **Interoperability**
  Predicate logic expressions can be used across NLP tools, semantic parsers, and database engines, providing a unified framework for knowledge representation.

I) Analyze an image series with multiple transformations and explain how to deduce the underlying rule.

II) Explain the process of grouping structurally similar geometric figures using Hu Moments for feature extraction and DBSCAN for clustering.

I) Role of Transformations in Image Series Problems

Transformations refer to consistent changes or manipulations applied to image elements across the series. These can be:

- **Geometric transformations**: rotation, reflection, scaling, translation.
- **Attribute changes**: change in color, shape, number, size, or texture.
- **Positional changes**: movement of elements in a grid or along a direction.
- **Combinatorial operations**: merging, splitting, or overlapping of shapes.

Example:
If a square rotates 90° clockwise in each image, the transformation is a **geometric rotation**.
Images: ■ → ◢ → ◣ → ◰ → ?

The correct answer would involve continuing this rotational transformation.

Role of Rule Deduction in Image Series Problems

Rule deduction involves identifying and applying logical or mathematical rules that explain the transformations. It includes:

- **Identifying patterns**: Recognizing repeated sequences or relationships.
- **Establishing rules**: Forming generalized statements about how transformations occur.
- **Inference**: Predicting the next image based on derived rules.

Example:
If the number of circles increases by one in each image (1 → 2 → 3...), the rule is **increment by one**, and the deduction is that the next image will contain 4 circles.

Rule deduction is critical because transformations may be complex or involve multiple attributes, and only logical reasoning can untangle them.

Illustrative Example

Image Series:

1st image: One triangle at the top
2nd image: Two triangles, one at the top, one at bottom
3rd image: Three triangles forming a triangle shape
4th image: ?

Analysis:

- **Transformation**: Addition of one triangle per image
- **Rule Deduction**: Triangles are being added to form larger geometric patterns (e.g., a bigger triangle).
- **Prediction**: The 4th image might show four triangles arranged in a square or forming a larger triangle.

II) Feature Extraction Using Hu Moments

Hu Moments are a set of 7 shape descriptors derived from **image moments** that are **invariant** to **scale, translation, and rotation**. This makes them ideal for comparing geometric figures based on structure, not exact pixel values.

Steps:

- Convert the image to grayscale and apply **thresholding** or **edge detection** (e.g., using Canny) to isolate the figure.
- Calculate the **spatial moments** and **central moments** of the binary shape.
- Derive the **Hu Moments** (7 values) from these central moments.
- Each figure is now represented as a 7-dimensional vector that uniquely describes its shape structure.

Why Hu Moments?

- They encode global geometric properties and are **invariant to orientation, position, and size**—key in non-verbal reasoning where figures may appear rotated or resized.
- They capture structural information like symmetry, curvature, and contour layout.

Example: A rotated square and an upright square will have nearly identical Hu Moment vectors.

Clustering Using DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

Once feature vectors (Hu Moments) are extracted, clustering groups structurally similar figures.

Steps:

- Feed the Hu Moment vectors into the DBSCAN algorithm.
- Set two parameters: ε (epsilon, the neighborhood radius) and minPts (minimum points to form a cluster).
- DBSCAN forms clusters of points (shapes) that lie within dense regions and labels sparse/noisy points as outliers.

Why DBSCAN?

- Unlike K-Means, DBSCAN does **not require the number of clusters** to be specified in advance.
- It can identify **arbitrary-shaped clusters** and **handle noise/outliers**, which are common in visual data.
- Works well when shape distributions are non-uniform and unsupervised.

Example: It can separate triangles, circles, and squares without needing to specify "3 clusters" beforehand, and ignore distorted or irrelevant figures.